Kapitel 1
Eine kurze Geschichte der Windows-Anwendungsentwicklung

Inhalt

Die Einführung von Windows 8 ist der größte Sprung in der gesamten Lebenszeit dieses Betriebssystems. Es handelt sich nicht einfach um eine neue Version, die alte Fähigkeiten auf den neuesten Stand bringt und um gewünschte Funktionen erweitert, sondern – laut Aussage von Microsoft – um eine Neukonzeption des Betriebssystems. Ohne die Vorgeschichte von Windows 8 zu kennen, ist der Paradigmenwechsel nur schwer zu verstehen.

In diesem Kapitel lernen Sie zunächst die Geschichte des Betriebssystems seit seiner Einführung 1985 kennen. Dann erfahren Sie, wie sich die Entwicklungsverfahren und Werkzeuge parallel zu Windows entwickelt haben.

1.1 Das Leben von Windows

Die allererste Version von Microsoft Windows wurde am 20. November 1985 veröffentlicht. Sie war ausdrücklich für grafische Benutzerschnittstellen (kurz *GUIs* für *Graphical User Interface*) konzipiert worden und wurde von Microsoft Win-

dows als Zusatzkomponente für das eigene MS-DOS-Betriebssystem angeboten. Windows wandelte den Massenmarkt für Personal Computer grundlegend. Die erste Version von Windows verwendete sehr einfache Grafiken und war eher ein Frontend für MS-DOS als ein echtes Betriebssystem.

1.1.1 Von Windows 3.1 zu 32-Bit

Nach der Veröffentlichung der ersten Version vergingen fast sieben Jahre, bevor Windows 3.1 im März 1992 eingeführt wurde. Dieses 16-Bit-Betriebssystem ermöglichte Multitasking in einer Umgebung, in der Benutzer diese Fähigkeit nicht erwarteten. Die neue Version von Windows enthielt virtuelle Gerätetreiber, die von mehreren DOS-Anwendungen gemeinsam verwendet werden. Da Windows 3.1 im *protected*-Modus arbeitete, konnte es mehrere Megabyte Speicher adressieren, ohne dass Software für virtuelles Speichermanagement erforderlich war. Im Vergleich dazu ließ der Standardadressmodus der 8086-CPU-Familie damals nur 640 KB zu. Ältere Computeranwender erinnern sich vielleicht noch an den Begrüßungsbildschirm dieses Betriebssystems (siehe Abbildung 1.1).

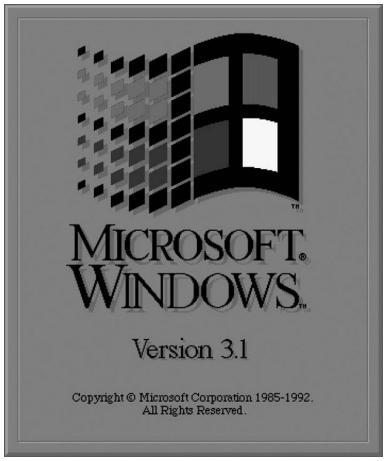

Abbildung 1.1 Der Begrüßungsbildschirm von Windows 3.1

Hinweis

Multitasking ist die Fähigkeit von Betriebssystemen, gleichzeitig mehreren Aufgaben (so genannte *Prozesse*) auszuführen. Diese Aufgaben nutzen die Ressourcen des Computers, wie etwa die CPU und den Speicher, gemeinsam. Das Betriebssystem wechselt bei der Ausführung von einer Aufgabe zur nächsten und führt jede Aufgabe für eine kurze Zeitspanne aus. Der schnelle Wechsel erweckt den Eindruck einer simultanen Ausführung.

Der *protected*-Modus ist ein Arbeitsmodus von x86-kompatiblen CPUs, in dem besondere Funktionen wie etwa die virtuelle Speicherverwaltung oder Multitasking genutzt werden können.

Im August 1995 wurde Windows 95 veröffentlicht, eine 32-Bit-Version des Betriebssystems, die Preemptive Multitasking unterstützte. Anders ausgedrückt: Das Betriebssystem konnte eine Aufgabe ohne aktive Beteiligung der Aufgabe unterbrechen. Windows 95 war kein Zusatz für MS-DOS mehr, sondern ein ausgewachsenes Betriebssystem (eine Tatsache, über die lange Zeit gestritten wurde). Es folgten einige weitere Windows-Versionen, insbesondere Windows 98 und Windows Me, bis im Oktober 2001 Windows XP veröffentlicht wurde.

1.1.2 Windows XP und Windows Vista

Abbildung 1.2 Das Windows-XP-Logo

Windows XP mit seinem berühmten Logo (siehe Abbildung 1.2) entwickelte sich zur beliebtesten Version von Windows. Seltsamerweise war dieser Erfolg (gemessen an der riesigen Anzahl von Installationen) nur zum Teil auf seine neue Benutzererfahrung (XP steht für »eXPerience«, deutsch »Erfahrung«) zurückzuführen. Auch andere Innovationen wie etwa das grafische Subsystem GDI+, der schnelle Benutzerwechsel, die *ClearType*-Schriftdarstellung, die 64-Bit-Unterstützung und viele andere waren ebenfalls nur teilweise für den Erfolg dieser Version von Windows verantwortlich. Tatsächlich war die Hauptursache für den Erfolg von Windows XP die Unbeliebtheit von Windows Vista, seinem Nachfolger.

Windows Vista wurde im November 2006 veröffentlicht. Es hatte ein brandneues Design und bot eine wesentlich verbesserte Sicherheit. Im Gegensatz dazu benötigte Windows XP drei Service Packs, um Sicherheitslücken und -probleme zu beseitigen. Dies hätte ausreichen können, Vista beliebter als seinen Vorgänger zu machen. Doch die neuen Funktionen von Vista erforderten eine leistungsstärkere Hardware. Die meisten Unternehmen hatten einen großen Teil ihres IT-Budgets für die Stabilisierung von XP ausgegeben. Nach dem Windows-XP-Service-Pack-3 (SP3) schien ein Umstieg auf Vista einfach nicht vernünftig zu sein. So wurde aus Vista schnell das kurzlebigste Betriebssystem in der Windows-Familie.

1.1.3 Windows 7 bügelt das Vista-Fiasko aus

Steven Sinofsky (Leiter der Windows-Entwicklungsabteilung von Microsoft) hat mehrfach gestanden, Microsoft habe aus seinen Fehlern gelernt, bevor es mit der

Entwicklung von Windows 7 begann, das im Juli 2009, zwei Jahre und acht Monate nach Vista veröffentlicht wurde. Die Performance von Windows 7 wurde im Vergleich zu Windows XP und Vista beträchtlich verbessert. So wurden etwa die Zeiten für das Hoch- und Runterfahren verkürzt, die Aufgabenplanung auf Mehrkern-CPUs (CPUs mit mehr als einem Prozessor) verbessert sowie die Suche und vieles mehr optimiert. In Windows 7 wurde die Benutzererfahrung durch neue Funktionen wie etwa Sprunglisten, Aero Peek, Aero Snap und viele andere kleine Dinge erheblich verbessert, die die Navigation zwischen Anwendungsfenstern sowie ihre Anordnung auf dem Bildschirm erleichterten.

Durch Windows 7 wurde das Vista-Fiasko erfolgreich ausgebügelt. Deshalb wäre es für das Windows-Team leicht gewesen, der von Windows 7 vorgegebenen Richtung zu folgen; doch stattdessen widmete es sich dem dringendsten Problem.

1.1.4 Der Paradigmenwechsel von Windows 8

Obwohl Windows in einer Ära geboren wurde, in der Personal Computer Einzug in den Alltag gehalten hatten, war es immer noch ein Betriebssystem, das im Hinblick auf Unternehmen und Informationsarbeiter konzipiert worden war. Die meisten von dem Betriebssystem angebotenen Funktionen drängten oder zwangen den Benutzer zu Arbeitsabläufen, die fest in das System eingebaut waren. Benutzer konnten Aktionen erst ausführen, wenn sie sich mit Konzepten wie etwa Dateien, Verzeichnissen, Sicherheitsgruppen, Berechtigungen, Shares, der Registry und so weiter vertraut gemacht hatten. Dieser Ansatz spiegelt wider, wie Benutzer nach Ansicht der Designer mit dem System interagieren sollten.

Hinweis

Weil Anwendungen die verfügbaren Systemressourcen unterschiedlich nutzten, musste das Windows-Betriebssystem regelmäßig gewartet werden: Cleanups (Bereinigungen), Festplattendefragmentierung, Virenprüfung, Service-Pack-Installationen und so weiter gehörten zur Arbeit mit Windows. Obwohl jede neue Version diese Lasten durch nützliche Verbesserungen erleichterte oder automatisierte, konnten sie nie vollkommen beseitigt werden.

Microsoft macht die ersten Schritte auf die Konsumenten zu

Die Konsumentenprodukte von Apple wie etwa das iPhone oder das iPad haben bewiesen, dass man auch intuitiver mit Computer-Software interagieren konnte, ohne zu wissen, was eine Datei, ein Verzeichnis, eine System-Registry oder eine Anwendungsinstallationsprozedur war. Microsoft schien sich diesem Ansatz lange

zu verschließen, aber die Umsatzzahlen des Marktes zwangen das Unternehmen, sich stärker auf Geräte und Betriebssysteme zu konzentrieren, bei denen der Konsument im Mittelpunkt stand.

Der erste ernsthafte Wandel im Verhalten von Microsoft wurde Mitte Februar 2010 auf dem *Mobile World Congress* in Barcelona, Spanien, erkennbar, als das Unternehmen zum ersten Mal Windows Phone 7 der Öffentlichkeit vorstellte. Bei Windows Phone 7 stand die Benutzererfahrung im Mittelpunkt. Das *User Interface* (*UI*, deutsch die *Benutzerschnittstelle*) bestach durch sein visuelles Design, seine Einfachheit und seine sanften, natürlich wirkenden Animationen, die eine sehr intuitive Bedienung dieses Geräts ermöglichten. Der Markt pries den Wandel; und jetzt, fast ein Jahr nach der »Mango«-Veröffentlichung von Windows Phone 7.5, ist Microsoft der dritte große Player im Markt für mobile Betriebssysteme und nähert sich iOS von Apple und Android Google immer mehr an.

Hinweis

Bereits vor Windows Phone 7 umfasste die Windows-Betriebssystemfamilie Versionen für eingebettete Geräte (Windows Embedded) und Mobilgeräte (Windows-CE und Windows-Mobile).

Windows 8 erscheint auf der Szene

Der konsumentenzentrische Ansatz von Windows Phone 7 wurde in Windows 8 übernommen. Die neue Startseite von Windows 8 (dessen Startzeit im Vergleich zu Windows 7 beträchtlich verkürzt wurde) erinnert nicht an den früheren Desktop mit seiner Taskleiste am unteren Rand. Stattdessen präsentiert Ihnen Windows 8 eine Reihe von »Kacheln«, die jeweils eine Anwendung repräsentieren (siehe Abbildung 1.3).

Diese neue Startseite sendet eine klare Botschaft an Konsumenten. Windows ist nicht nur ein Betriebssystem für Informationsarbeiter und erfahrene Computeranwender, sondern auch ein Produkt für Konsumenten, das für die Bedienung per Fingereingabe, für Tablets und ähnliche Geräte konzipiert wurde. Die Oberfläche der Startseite ist sehr intuitiv; und die meisten Menschen können ohne Einweisung sofort anfangen, herumzuspielen. Benutzer, die Erfahrungen mit den Touch-Bildschirmen ihrer Smartphones und Tablets haben, kommen spontan mit der Startseite, dem Start von Anwendungen, dem Scrollen, Zoomen und der Anwendung von Gesten zurecht, die sie bereits mit anderen Geräten gelernt haben.

Abbildung 1.3 Die Windows-8-Startseite

Für Anwender, die unter Windows mit Geschäftsanwendungen (wie etwa Word, Excel oder PowerPoint) oder anderen unternehmensspezifischen UIs arbeiten, mag die Änderung der Windows-UI ungewöhnlich erscheinen. Sie können beruhigt sein, denn Windows 8 ist mit vorhandenen Anwendungen vollkommen kompatibel und verfügt auch über einen Desktop-Modus. Wenn Sie eine Anwendung starten, die für frühere Windows-Versionen geschrieben wurde (oder die einfach nicht die Windows-8-Style-UI verwendet), wird die Anwendung in der wohlbekannten Desktop-Umgebung ausgeführt. Startet ein Benutzer etwa Excel, wird dieses Programm im Desktop-Modus geöffnet (siehe Abbildung 1.4).

Dies ist das zweite Gesicht von Windows 8. Es ist allen vertraut, die bereits früher mit Windows gearbeitet haben. Wenn Sie nicht auf das Start-Menü und den Statusindikator achten, der Zeit und Datum anzeigt, könnten Sie meinen, mit Windows 7 zu arbeiten.

Die neue Windows-8-Startseite ist nicht einfach ein Zusatz zu Windows 7. Hinter diesem einfachen Bildschirm verbirgt sich eine brandneue Welt konsumentenzentrierter Anwendungen, die als *Windows-8-Style-Anwendungen* bezeichnet werden. Benutzer betrachten nicht einen Desktop voller Symbole und rechteckiger Fenster mit Anwendungen, sondern immer nur eine Anwendung auf einmal, die den gesamten Bildschirm einnimmt. Keine Titelleiste, keine Schaltfläche zum Schließen des Fensters, keinen Rahmen zur Größenveränderung und keine anderen Elemente (so genannter »Zierrat« oder »Chrome« in der User-Experience-Terminologie) lenken

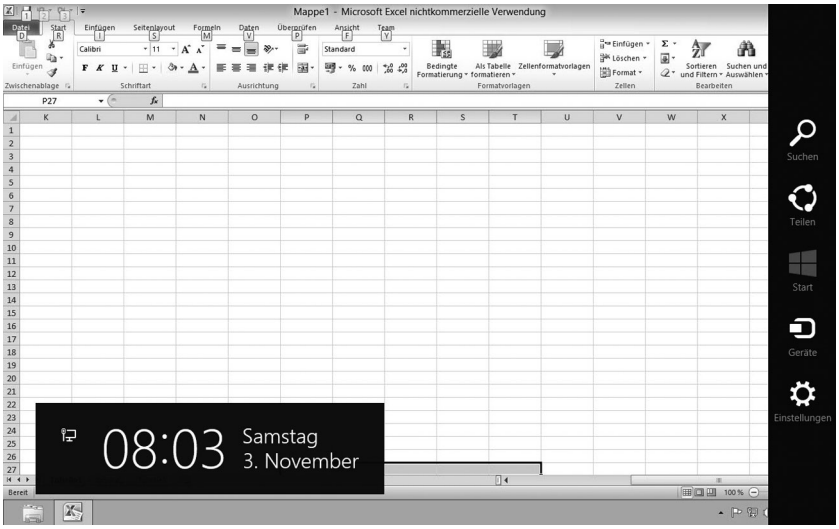

Abbildung 1.4 Excel im Desktop-Modus von Windows 8

den Blick des Benutzers von der Anwendung ab. Die WETTER-Anwendung ist ein großartiges Beispiel für diesen neuen Stil (siehe Abbildung 1.5).

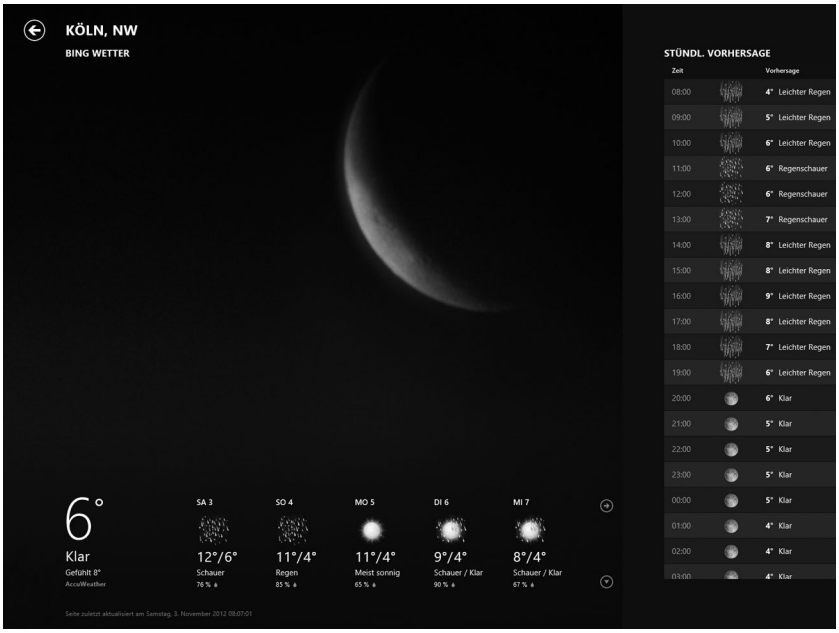

Abbildung 1.5 Die Wetter-Anwendung in Windows 8

In diesem Buch lernen Sie dieses neue UI-Paradigma von Grund auf kennen. Wichtiger ist jedoch, dass Sie auch lernen, solche Anwendungen zu entwickeln. Doch bevor Sie tiefer in die Entwicklung von Windows-8-Style-Anwendungen einsteigen, sollten Sie kurz die Geschichte der Windows-Entwicklung rekapitulieren.

1.2 Geschichte der APIs und Werkzeuge

Ohne Anwendungen, die auf dieser Plattform laufen, und Entwickler, die diese Programme geschrieben haben, wäre die Geschichte von Windows ziemlich nichtssagend. Als Unternehmen hat sich Microsoft stets darum bemüht, um seine Produkte herum eine starke Entwickler-Community aufzubauen und zu pflegen. Dies gilt auch für sein Flaggschiff Windows.

Tipp

Die Bedeutung der Entwickler-Community wird von führenden Microsoft-Mitarbeitern immer wieder unterstrichen. Wenn Sie im Internet nach »Steve Ballmer« suchen, stoßen Sie unweigerlich auch auf das Video, in dem er leidenschaftlich »Entwickler, Entwickler, Entwickler, …« ruft und das Wort ein Dutzend Mal wiederholt.

2012 wurde die Windows-Plattform 27 Jahre alt. In diesen vielen Jahren wurden nicht nur das Betriebssystem, sondern auch seine *APIs* (*Application Programming Interfaces*) und die zugehörigen Entwicklungswerkzeuge erheblich verbessert. Doch mit Version 8 macht Windows den größten Sprung in seiner gesamten Geschichte. Ein Blick in die Vergangenheit, und zwar genau auf die allerersten Momente der Windows-Anwendungsentwicklung, hilft verstehen, warum dies so ist.

1.2.1 Die Leistungsstärke von C

Früher wurden längst nicht so viele Windows-Anwendungen programmiert wie heute. Damals betrachteten Programmierer, die mit MS-DOS-Anwendungen aufgewachsen waren, den Windows-Ansatz als verdreht, als wäre das Innere nach außen gekehrt worden. Auch wenn eine saubere MS-DOS-Anwendung alles kontrollierte, rief sie bei Bedarf Funktionen des Betriebssystems auf. Nicht so Windows! Das Betriebssystem kontrollierte die Anwendung und rief sie auf, wenn das Programm eine Aufgabe erledigen sollte, etwa um die UI aufzufrischen oder einen Menübefehl auszuführen.

Und dies war nicht die einzige Gräueltat, die den armen Entwicklern angetan wurde. Mit der damals führenden Programmiersprache C die einfachste »Hallo Welt«-Anwendung in MS-DOS zu schreiben, war ein Kinderspiel. Ein Beispiel:

```
#include <stdio.h>

main()
{
    printf("Hallo Welt");
}
```

Um dasselbe Ergebnis unter Windows zu erzielen, war beträchtlich mehr Aufwand erforderlich. Der Entwickler musste so genannten »Scaffolding«-Code schreiben, um die einzige `printf`-Funktion einzuhüllen. Der Vorgang war alles andere als intuitiv (siehe Listing 1.1).

```
#include <windows.h>

/* Export des Einstiegspunktes, der von Windows aufgerufen werden soll
*/
long FAR PASCAL _export WndProc(HWND, UINT, UINT, LONG)

/* Der Einstiegspunkt der Anwendung */
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
    LPSTR lpszCmdParam, int nCmdShow)
{
    static char szApplication[] = "HalloW";
    HWND hwnd;
    MSG msg;
    WNDCLASS wndClass;

    /* Eine Window-Klasse erstellen */
    if (!hPrevInstance)
    {
      wndClass.Style = CS_HREDRAW | CS_VREDRAW;

      wndClass.lpfnWndProc = WndProc;
      /* Der Kürze halber wurden einige Zeilen ausgelassen. */
      wndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
      wndClass.lpszMenuName = NULL;
      wndClass.lpszClassName = szApplication;
      RegisterClass(&wndClass);
    }

    /* Eine Instanz der Window-Klasse erstellen */
    hwnd = CreateWindow(szApplication,
      "Mein Hallo-Welt-Programm",
```

```
      WS_OVERLAPPEDWINDOW,
      /* Der Kürze halber wurden einige Parameter ausgelassen. */
      hInstance,
      NULL);
    /* Die Anzeige des Fensters initiieren */
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    /* Die Message Loop verwalten */
    while (GetMessage(&msg, NULL, 0, 0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg)
    }
}

/* Messages verarbeiten */
long FAR PASCAL _export WndProc(HWND hwnd, UINT message,
    UINT wParam, LONG lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;

    switch (message)
    {
      case WM_PAINT:
        hdc = BeginPaint(hwnd, &ps);
        GetClientRect(hwnd, &rect);
        DrawText(hdc, "Hallo Welt", -1, &rect,
            DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        EndPaint(hwnd, &pd);
        return 0;

      case WM_DESTROY:
        PostQuitMessage(0);
        return 0
     }
     return DefWindowProc(hwnd, message, wParam, lParam);
}
```

Listing 1.1 Das »Hallo Welt«-Programm à la Windows 3.1 (Auszug)

Dieses Programm ist so umfangreich, weil die Windows-API nur systemnahe Betriebssystemfunktionen anbot. Dieser lange Quellcode zeigt aber wichtige in-

terne Details von Windows, die sich auch heute noch – natürlich in verbesserter Form – im Kern von Windows 8 befinden:

- Ganz am Anfang erstellt das Programm eine Window-Klasse, indem es mit der Methode `RegisterClass` die Felder der `wnd`-Klassenstruktur initialisiert. Die *Window-Klasse* identifiziert eine Prozedur (die so genannte *Window-Prozedur*), die Messages verarbeitet, die an das Fenster gesendet werden.
- Das Programm erstellt mit der Methode `CreateWindow` ein Fenster und verwendet dabei die registrierte Window-Klasse. Dann zeigt es das Fenster mit der Methode `ShowWindow` an. Die Methode `UpdateWindow` sendet eine Message an das Fenster, um sein UI neu darzustellen.
- Die Message Loop ist die Seele der Anwendung:

  ```
  while (GetMessage(&msg, NULL, 0, 0))
  {
      TranslateMessage(&msg);
      DispatchMessage(&msg)
  }
  ```

  Diese Schleife erhält Messages von einer Queue, übersetzt Tastenanschläge in entsprechende Messages (zum Beispiel ob die Maus verwendet wurde) und leitet sie dann an die zuständige Fensterprozedur weiter.
- Wenn die Message Loop die Seele ist, ist die Fensterprozedur das Herz. In Listing 1.1 wird `WndProc` von der Message Loop aufgerufen. Ihr `message`-Parameter enthält den Code der Message (das zu verarbeitende Ereignis); und eine `switch`-Anweisung hüllt die Code-Snippets ein, die einzelne Messages verarbeiten.
- Die `WM_PAINT`-Message weist das Fenster an, sich selbst neu darzustellen. Mit der `BeginPaint`-Methode fordert es eine Device-Kontextressource an, die es für das Zeichnen im Client-Bereich des Fensters verwendet. Mit diesem Device-Kontext wird dann die » Hallo Welt«-Meldung in der Mitte des Fensters dargestellt. `ReleasePaint` gibt den Device-Kontext wieder frei, der im System eine sehr knappe Ressource ist.

Sie können sich vorstellen, wie aufwendig und mühsam die Windows-Entwicklung damals war, weil Programmierer durch das Windows-API gezwungen waren, systemnahe Betriebssystemkonstrukte zu verwenden.

1.2.2 C++ verdrängt C

1983, nur wenige Jahre nachdem Brian Kernighan und Dennis Ritchie die erste Version von C (1978) veröffentlicht hatten, entwickelte Bjarne Stroustrup eine

neue Sprache, die C um objektorientierte Aspekte erweiterte. Diese Sprache war *C++*, die sich auch auf der Windows-Plattform schnell verbreitete.

In C++ können Daten und Funktionen in Klassen eingekapselt werden. Außerdem unterstützt die Sprache Vererbung und Polymorphismus. Damit konnte die flache API von Windows durch einen kleineren Satz von Entities repräsentiert werden, die Datenstrukturen und API-Operationen zu einem logischen Kontext zusammenfassten. Beispielsweise konnten alle Operationen, mit denen Fenster in der UI erstellt, angezeigt und verwaltet wurden, in eine Klasse namens `Window` eingeschlossen werden.

Der C++-Ansatz verschaffte Entwicklern einen besseren Überblick über die API und verringerte die Einstiegshürde in die Windows-Programmierung. Beispielsweise können die wesentlichen Teile des »Hallo Welt«-Programm aus Listing 1.1 in Objekten zusammengefasst werden (siehe Listing 1.2).

```
// --- Die Klasse für das Hauptprogramm
class Main
{
    public:
      static HINSTANCE hInstance;
      static HINSTANCE hPrevInstance;
      static int nCmdShow;
      static int MessageLoop(void);
};

// --- Die Klasse für ein Fenster
class Window
{
    protected:
      HWND hWnd;
    public:
      HWND GetHandle(void) { return hWnd; }
      BOOL Show(int nCmdShow) { return ShowWindow(hWnd, nCmdShow); }
      void Update(void) { UpdateWindow(hWnd); }
      virtual LRESULT WndProc(UINT iMessage, WPARAM wParam,
                              LPARAM lPa ram) = 0;
};

// --- Die Klasse für das Hauptfenster dieses Programms
class MainWindow : public Window
{
    // --- Die Implementierung wurde der Kürze halber weggelassen
};
```

```
// --- Auszug aus der Implementierung der Main-Klasse
int Main::MessageLoop(void)
{
    MSG msg;

    while(GetMessage((LPMSG) &msg, NULL, 0, 0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
    return msg.wParam;
}

LRESULT MainWindow::WndProc(UINT iMessage, WPARAM wParam,
                            LPARAM lParam)
{
    switch (iMessage)
    {
      case WM_CREATE:
        break;
      case WM_PAINT:
        Paint();
        break;
      case WM_DESTROY:
        PostQuitMessage(0);
        break;
      default:
        return DefWindowProc(hWnd, iMessage, wParam, lParam);
     }
     return 0;
}
```

Listing 1.2 Struktur des »Hallo Welt«-Programms in C++ (Auszug)

Mit dem objektorientierten Ansatz von C++ konnte das Verhalten von Objekten in wiederverwendbare Code-Bibliotheken gepackt werden. Mit diesen Bibliotheken konnten Programmierer eigene Programme schreiben, in denen sie nur noch das Verhalten definieren mussten, das von dem der eingebauten Objekte abwich. Beispielsweise mussten sie die `Paint`-Methode aus Listing 1.2 überschreiben, um die UI ihrer `Window`-Objekte neu darzustellen.

Mit C++ und Objekt-Bibliotheken wandelte sich die Windows-Programmierung erheblich. Microsoft entwickelte zwei Bibliotheken: die *Microsoft Foundation Classes*

(*MFC*) und die *Active Template Library* (*ATL*), die auch heute noch in *Visual Studio*, der wichtigsten Entwicklungsumgebung von Microsoft, angeboten und gepflegt werden. Sie lernen die Visual-Studio-Entwicklungsumgebung in Kürze näher kennen.

1.2.3 Visual Basic

Anwendungen, die in C oder C++ geschrieben waren, enthüllen zahlreiche Details über die Arbeitsweise von Windows. In einigen Fällen müssen Programmierer diese Dinge wissen, aber in den meisten Fällen lenken sie Entwickler davon ab, sich auf die eigentliche Anwendungsfunktionalität zu konzentrieren.

Mit Visual Basic, das im Mai 1991 veröffentlicht wurde, änderte sich dieser Programmierstil gravierend. Anstatt die internen Windows-Details offenzulegen, verbarg Visual Basic sie vor dem Programmierer und bot ihm stattdessen höhere, systemferne Konstrukte wie etwa Forms, Steuerelemente, Module, Klassen und Code-Behind-Dateien an. Anstatt Dutzende von Zeilen zu schreiben, um eine sehr einfache Funktionalität zu realisieren, konnten sich Entwickler in Visual Basic auf die Kernfunktionalität ihrer Anwendungen konzentrieren. Das »Hallo Welt«-Programm konnte in einer einzigen Zeile geschrieben werden:

```
MsgBox("Hallo Welt!")
```

Keine Einrichtung einer Window-Klasse, keine Window-Registrierung, keine Programmierung einer Message Loop! Die High-Level-Konzepte der Sprache machten den Umgang mit den Scaffolding-Details vollkommen überflüssig. Sie wurden alle von der Visual-Basic-Runtime implementiert.

Kombiniert mit einer grafischen IDE (Integrated Development Environment) ist diese Methode der Anwendungsentwicklung immer noch der beliebteste und produktivste Ansatz. Programmierer entwerfen grafisch die *Dialogfenster* (oder *Forms* in Visual-Basic-Terminologie) der Anwendung, indem sie UI-Elemente (Steuerelemente) aus der Toolbox der IDE auf die Oberfläche einer Form ziehen und ablegen. Jedes Steuerelement verfügt über mehrere Event-Handler, die auf die Ereignisse aus der Umgebung reagieren. Ein Ereignis ist etwa ein Klick des Benutzers auf eine Schaltfläche oder die Änderung der Auswahl in einem Kombinationsfeld. Das Programmieren besteht hauptsächlich darin, den Code zur Verarbeitung dieser Ereignisse zu schreiben.

1993 veröffentlichte Microsoft einen Binärstandard, das Component Object Model (COM) für wiederverwendbare Objekte, die von anderen Anwendungen benutzt werden konnten. Einige Technologien bauen auf COM auf, wie etwa Object Linking and Embedding (OLE), das eine Automatisierung von Anwendungen

ermöglichte. Die Versionen von Visual Basic, die nach 1993 veröffentlicht wurden, waren im Hinblick auf COM und OLE konzipiert worden. Dieser Ansatz war so erfolgreich, dass ein Dialekt der Sprache, *Visual Basic for Applications* (*VBA*), zur Programmiersprache von Microsoft-Office-Makros gewählt wurde.

1.2.4 Delphi

Visual Basic war nicht die einzige Programmierumgebung, die vom ausgetretenen Pfad von C und C++ abwich. Delphi, ursprünglich von Borland entwickelt, basierte auf der Programmiersprache Object Pascal. Während Visual Basic eine objektbasierte Sprache war (es unterstützte Klassen mit eingekapselten Daten und Funktionen, ermöglichte aber keine Objektvererbung), war Object Pascal eine echte objektorientierte Sprache. Die erste Version der IDE von Delphi wurde 1995 veröffentlicht. Sie ähnelte der IDE von Visual Basic sehr stark. Delphi war als Werkzeug für *Rapid Application Development* (*RAD*) konzipiert worden und unterstützte die Entwicklung einfacher und sogar unternehmenstüchtiger Datenbankanwendungen.

Delphi wurde sehr schnell weiterentwickelt. Allein in den ersten fünf Jahren seiner Existenz wurden fünf Versionen veröffentlicht. Delphi war das erste Werkzeug, das 32-Bit-Anwendungen für Windows kompilieren konnte. Vergleichbar mit den Visual-Basic-Steuerelementen bot Delphi mehr als hundert visuelle Komponenten an, die in der so genannten *Visual Component Library* (*VCL*) zusammengefasst waren und von Entwicklern sofort praktisch eingesetzt werden konnten. Außerdem konnten Entwickler leicht eigene visuelle Komponenten erstellen und in die vorhandene Bibliothek einfügen.

1.2.5 Die Einführung von .NET

2002 brachte das Microsoft .NET Framework frischen Schwung in die Windows-Entwicklung. .NET-Programme werden in eine Intermediate-Sprache, die so genannte *Microsoft Intermediate Language* (*MSIL*), kompiliert. Dieser Intermediate-Code wird von einem einfachen JIT-Compiler (Just-in-Time-Compiler) zur Laufzeit in ausführbare CPU-spezifische Operationen übersetzt. Dieser neue Ansatz führte mehrere wichtige Paradigmen in die Windows-Entwicklung ein:

- Vor der Einführung von .NET verwendete jede Sprache (und Entwicklungsumgebung) ihre eigene Runtime-Bibliothek. Wer eine neue Sprache lernte, musste auch das Arbeiten mit einer neuen Runtime-Bibliothek lernen. Bei .NET verwenden alle Sprachen dieselbe Runtime. 2002 unterstützte Microsoft nur zwei Programmiersprachen: C# und Visual Basic .NET. Heute werden mehr als

100 .NET-Sprachen verwendet. Microsoft selbst ergänzte diese Liste um F#. Außerdem unterstützt das Unternehmen IronPython und IronRuby (die von eigenen Communities entwickelt und gepflegt werden).

- Die Nutzung eines Garbage-Collection-Mechanismus: Die Laufzeitumgebung verwaltet die Speicherallokation und Objektzerstörung automatisch. Dieses Verhalten steigert die Produktivität und verringert die Fehleranfälligkeit von Code. Außerdem verringert die Garbage Collection die Gefahr, Programme mit Speicherlecks zu schreiben.
- Programmierer müssen nicht mehr mit systemnahen API-Methoden arbeiten, sondern können Objekte verwenden, die die Komplexität der APIs im Hintergrund verbergen. Anstatt sich mit kleinteiligen, internen Windows-Details herumzuschlagen, können Entwickler höhere Abstraktionen verwenden und so ihre Produktivität steigern.
- .NET erleichtert die Kooperation zwischen COM- und .NET-Objekten erheblich. .NET-Code kann nicht nur auf COM-Objekte zugreifen, sondern umgekehrt auch Objekte bereitstellen, die von der COM-Welt genutzt werden können.

.NET wird als *verwaltete Umgebung* bezeichnet und seine Sprachen werden *verwaltete Sprachen* genannt, um sie von nativen Sprachen wie etwa C, Object-Pascal (Delphi) oder C++ zu unterscheiden, die in CPU-spezifischen Code kompiliert werden.

Hinweis

Das Microsoft .NET Framework war nicht die erste verwaltete Laufzeitumgebung. Diese Auszeichnung gebührt Java, das 1995 von Sun Microsystems veröffentlicht wurde. .NET war Microsofts Antwort auf das Java-Phänomen; und eine Reihe seiner Funktionen waren von der Java-Implementierung von Sun inspiriert.

Visual Studio wurde parallel zu Microsoft .NET Framework veröffentlicht und war ein wichtiger Faktor für den Erfolg von .NET. Visual Studio wurde mit rund zwei Dutzend Projektvorlagen ausgeliefert, um den Start der Windows-Anwendungsentwicklung zu beschleunigen. Heute enthält die Ultimate-Ausgabe von Visual Studio mehr als hundert Projektvorlagen. Mehr über Visual Studio erfahren Sie in Kapitel 4.

Visual-Studio-Vorlagen sind sehr leistungsstark. So können Sie etwa die Anwendung in Abbildung 1.6 mit einer Windows-Forms-Anwendung-Vorlage mit wenigen Klicks erstellen.

Abbildung 1.6 Eine einfache, mit Visual Studio erstellte Windows-Forms-Anwendung

Mit den Werkzeugen von Visual Studio können Sie leicht die visuellen Eigenschaften des Fensters in Abbildung 1.6 festlegen. Listing 1.3 zeigt nur den Code, den Sie manuell zu diesem Beispiel hinzufügen müssen.

```
using System;
using System.Windows.Forms;

namespace HelloWorldWinForms
{
    public partial class Form1 : Form
    {
      public Form1()
      {
        InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
        label1.Text = "Hello from Windows Forms!";
      }
    }
}
```

Listing 1.3 Der Code hinter der Form aus Abbildung 1.6

Obwohl .NET mit reichhaltigen Objekt-Bibliotheken und großartigen Werkzeugen ausgestattet war, benutzte es immer noch die API, die auf dem Design der ersten Windows-Versionen basierte.

1.2.6 Neue UI-Technologien

Die Windows-UI wurde lange von der Graphics-Device-Interface-(GDI-)API verwaltet, das mit der Veröffentlichung von Windows XP zu GDI+ erweitert worden

war. GDI und GDI+ sind rasterbasierte APIs. Alle standardmäßigen Windows-Steuerelemente benutzten diese API, um sich selbst darzustellen. Entwickler konnten das Aussehen dieser Steuerelemente nur ändern, indem sie das Windows-Ereignis überschreiben, das die UI der Steuerelemente darstellte.

Als im Microsoft .NET Framework 3.0 mit *Windows Presentation Foundation (WPF)* ein neues grafisches Subsystem eingeführt und später als Komponente in Windows Vista integriert wurde, änderte sich das GDI-Paradigma vollkommen. Anstatt die UI imperativ zu erstellen (das heißt, einschlägige Anweisungen in einer Programmiersprache zu schreiben), beschreibt WPF UI-Elemente mit der Sprache *XAML (eXtensible Application Markup Language*), einem Derivat von XML (eXtensible Markup Language). Außerdem nutzt WPF die großartigen Fähigkeiten der Hardware-beschleunigten Darstellung, die von den eingebauten GPUs (Graphic Processing Units) der Computer angeboten werden.

Auch Silverlight, das reichhaltige Internet-Anwendungsframework von Microsoft, definiert seine Benutzerschnittstellen mit XAML. Listing 1.4 zeigt ein sehr einfaches XAML-Beispiel in Silverlight.

```
<UserControl x:Class="HelloFromSL.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/
                  2006"
    mc:Ignorable="d"
    d:DesignHeight="300"
    d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
      <TextBlock FontSize="48">Hello from Silverlight<;/TextBlock>
    </Grid>
</UserControl>
```

Listing 1.4 Die Datei MainPage.xaml der Siverlight-Anwendung

Dieser Code erzeugt die Seite aus Abbildung 1.7. Der Text in dieser Abbildung wird durch die hervorgehobene Codezeile in Listing 1.4 dargestellt.

WPF und Silverlight sind großartige Technologien. Mit ihnen zu arbeiten ist vergleichbar damit, einen Lamborghini-Motor aus den 1930er Jahren (GDI/GDI+) durch einen modernen zeitgenössischen Motor (WPF/Silverlight) auszutauschen. Diese Technologien definieren nicht nur die UI, sondern können auch deren dynamisches Verhalten mit wenig (oder ganz ohne) Code deklarieren. Diese Tech-

Abbildung 1.7 Der Output von Listing 1.4

nologien verbinden die UI mit der Logik- oder Modell-Schicht einer Anwendung. Die folgende Aufstellung enthält einige Eigenschaften dieser Technologien:

- Diese Technologien wurden für die Entwicklung leistungsstarker Desktop- oder Internet-Anwendungen mit einer überlegenen Benutzererfahrung konzipiert. Sie bieten nicht nur einfache UI-Elemente (wie etwa Textfelder, Schaltflächen, Listen, Kombinationsfelder, Bilder und so weiter) an, sondern ermöglichen auch die Einbindung von Inhalten mit *Animationen* und von *Medienelementen* wie etwa Video oder Audio. Im Gegensatz zu dem traditionellen (einige sagen sogar: langweiligen) UI-Ansatz mit rechteckigen UI-Elementen können Sie mit WPF und Silverlight das gesamte Aussehen einer Anwendung ändern.
- Sie stellen ein sehr flexibles *Layout-System* zur Verfügung, mit dem Sie leicht Layouts erstellen können, die sich automatisch an diverse Faktoren anpassen (wie etwa die verfügbare Bildschirmgröße, die Anzahl der angezeigten Elemente und deren Größe beziehungsweise Vergrößerung).
- *Stile* und *Vorlagen* fördern die reibungslose Zusammenarbeit zwischen Entwicklern und Designern. Entwickler implementieren die Anwendungslogik und setzen die visuellen Eigenschaften der UI niemals direkt. Stattdessen signalisieren sie per Programm Zustandsänderungen der UI. Designer erstellen die visuellen UI-Elemente, wobei sie die möglichen Zustände der UI berücksichtigen.
- Silverlight und WPF verbinden Daten und UI-Elemente deklarativ (das heißt ohne Code). Die Datenbindung funktioniert bei Stilen, Vorlagen, Layouts und sogar Animationen. Dieser Mechanismus ist besonders bei branchenspezifischen Anwendungen nützlich. Mit Hilfe von Datenbindungsinformationen aus der Datenbank und der Anwendungslogik können Elemente deklarativ an die UI-Elemente gebunden werden.

Hinweis

Ein Layout-System dient zur Anordnung von Elementen in der UI. Mit einem flexiblen Layout-System (wie etwa hinter WPF, Silverlight und Windows 8) können Sie Beziehungen zwischen UI-Elementen definieren, die zur Laufzeit automatisch in der UI angezeigt werden. So können Sie etwa im Layout-System definieren, dass Steuerelemente in zwei Spalten angeordnet und gleichmäßig verteilt werden sollen.

Stile und Vorlagen werden in WPF und Silverlight ähnlich wie Stile und Vorlagen in Satzsystemen verwendet. Ein Stil fasst gemeinsame Attribute von UI-Elementen zusammen; eine Vorlage ist ein Gerüst mit Platzhaltern für die Inhalte eines UI-Elements.

Ein Hauptnutzen von WPF und Silverlight besteht wahrscheinlich darin, dass sie Ihnen helfen können, UI-bezogene Aufgaben eines Entwicklers sauber von den Aufgaben eines Designers zu trennen. Bei den heutigen konsumentenzentrischen Anwendungen ist dies offensichtlich vorteilhafter als Ansätze, bei denen die Aufgaben von Entwicklern und Designern miteinander verwoben sind.

1.3 Fallstricke der Windows-Anwendungsentwicklung

Beim Ausbau der Windows-Anwendungsentwicklung zweigten Technologien und zugehörige Techniken ab. Ein Zweig widmete sich der nativen Entwicklung, die mit der C-Programmiersprache in den Anfangstagen von Windows begann. Der andere Zweig bevorzugte die verwaltete Entwicklung mit dem Microsoft .NET Framework und den einschlägigen verwalteten Technologien und Sprachen.

Weil native Anwendungen CPU-spezifisch kompiliert werden und systemnah auf die Windows-API und Systemressourcen zugreifen, können sie eine großartige Performance bieten, an die verwaltete Technologien nicht heranreichen. Doch Geschäftsanwendungen mit nativem Code zu schreiben, ist weniger produktiv und umständlicher als mit verwaltetem Code mit dem Microsoft .NET Framework. Da Geschäftsanwendungen im Allgemeinen mit Datenbanken arbeiten, ist der Performance-Overhead von verwalteten Sprachen gar nicht wahrnehmbar; denn die Anwendung kommuniziert hauptsächlich mit zugrunde liegenden Diensten und Datenbanken.

Windows-Anwendungsentwickler fragen sich natürlich, welches die richtige Technologie und das richtige Werkzeug für eine bestimmte Anwendung ist. Oft stecken sie dabei in einer Zwickmühle.

Häufig, insbesondere bei der Entwicklung von Desktopanwendungen, gibt es keine optimale Wahl zwischen nativer und verwalteter Entwicklung. Mit WPF und Silverlight kann man zwar spektakuläre Anwendungen entwickeln, aber ein Zugriff von nativem Code ist nicht möglich, weil WPF und Silverlight nicht tief genug in das Betriebssystem eingebettet sind. So dauert es eine gewisse Zeit, eine WPF-Anwendung zu starten, während das WPF-Subsystem in den Speicher geladen wird. Bei vielen rechenintensiven Anwendungen führt .NET zu schlechteren Ergebnissen als nativer Code.

Visual Studio ist ein großartiges Beispiel dieser schizophrenen Situation. Es arbeitet mit einer gemischten Codebasis. Die meisten Komponenten sind in C++ implementiert und andere Teile in C# (mit Hilfe von WPF). Der Begrüßungsbildschirm sollte sofort auf dem Bildschirm erscheinen, wenn der Benutzer die Anwendung startet. Deshalb ist der Bildschirm in C++ mit GDI+ implementiert, weil WPF nicht schnell genug gewesen wäre. Der Code-Editor von Visual Studio ist in WPF implementiert, weil er über einen reichhaltigen und benutzerfreundlichen Werkzeugsatz für das Hosten einer solchen komplexen Komponente mit einer großartigen UI verfügt.

Bei der Entwicklung von Branchenanwendungen ist Produktivität der wichtigste Faktor. Die möglicherweise bessere Performance von nativem Code ist für solche Anwendungen kein echter Mehrwert, weil meistens die Datenbank den Performance-Engpass bildet. Geschäftsanwendungen können im Allgemeinen mit verwaltetem Code viel schneller und mit weniger Fehlern programmiert werden.

Das Dilemma: Wenn Sie eine Desktopanwendung schreiben, können Sie mit nativem Code eine bessere Performance erzielen, dann aber nicht den reichhaltigen Werkzeugsatz von WPF und Silverlight verwenden. Mit verwaltetem Code werden Sie produktiver, aber auf Kosten der besseren Performance. Entscheiden Sie sich aus schierer Verzweiflung wie Visual Studio für eine Zwitterlösung, haben Sie einen erheblichen Overhead. Keine leichte Entscheidung, nicht wahr?

Mit Windows 8 ändert sich diese Situation total. Webentwickler mit HTML5-/CSS3-/JavaScript-Erfahrung, gestandene C++-Programmierer sowie Entwickler, die mit .NET groß geworden sind, können alle sicher sein, dass ihr Wissen ausreicht, um in die Windows-8-Style-Anwendungsentwicklung einzusteigen. Es gibt kein privilegiertes Lager von Geeks, wie etwa das der C-Programmierer am Beginn der Windows-Ära. Jeder kann dieselben Technologien und dieselben Werkzeuge verwenden, ohne Kompromisse einzugehen. Wie ist dies möglich? Die Erklärung finden Sie in Kapitel 3.

1.4 Zusammenfassung

Die Windows-Plattform hat sich seit 1985 erstaunlich entwickelt. Aus einer einfachen MS-DOS-Erweiterung ist ein komplexes, ausgewachsenes Betriebssystem geworden, das auf der Mehrzahl der Personalcomputer auf der ganzen Welt läuft. Windows wurde für Informationsarbeiter und erfahrene Computeranwender konzipiert. Obwohl es sich mit jeder Version immer stärker auch weniger erfahrenen Benutzern annäherte, konnte es niemals das Betriebssystem für solch einfache Consumer-Geräte wie etwa das iPhone oder das iPad von Apple werden.

Windows 8 ändert dies. Zusätzlich zum traditionellen Desktop-Modus, mit dem alle Windows-Benutzer vertraut sind, hat das Betriebssystem mit den Windows-8-Style-Anwendungen ein neues Gesicht bekommen, das eine intuitive Benutzererfahrung vermittelt. Die neuen Anwendungen beanspruchen jeweils den kompletten Bildschirm und die volle Aufmerksamkeit des Benutzers.

Doch nicht nur das Betriebssystem, sondern auch die Entwicklungswerkzeuge und APIs haben sich geändert und wurden erheblich verbessert. Im Gegensatz zu früher können nicht nur C- und C++-Programmierer Windows-Anwendungen erstellen. Heute können alle C++-, Visual-Basic-, C#- oder Delphi-Programmierer mit ihren bevorzugten IDEs ebenfalls Anwendungen entwickeln. Je nach Sprache und Werkzeug müssen Programmierer verschiedene APIs und Technologien verwenden. Während native Programmiersprachen (wie etwa C, C++ oder Delphi) CPU-spezifischen Code generieren und eine bessere Performance liefern, können Sie Branchenanwendungen mit verwaltetem Code produktiver entwickeln. Außerdem können Sie leicht auf moderne UI-Technologien wie etwa WPF und Silverlight zugreifen.

Windows-8-Style-Anwendungen arbeiten sowohl bei nativem Code als auch bei verwaltetem Code kompromisslos mit derselben API. Darüber hinaus steht diese API auch für HTML5-/CSS3-/JavaScript-Entwickler zur Verfügung.

Windows 8 ändert nicht nur die UI des Betriebssystems, sondern auch die Interaktion mit ihm. Es unterstützt nicht nur traditionelle Eingabegeräte wie Tastatur oder Maus, sondern stellt auch eine erstklassige Unterstützung für die Interaktion mit dem Betriebssystem und Windows-8-Style-Anwendungen zur Verfügung, einschließlich der Interaktion mit Multi-Fingereingabegesten, einem Zeichenstift oder durch Sensoren wie etwa Gyroskop und Beschleunigungssensor. In Kapitel 2 lernen Sie diese neuen Dinge für das Arbeiten mit Windows kennen.

Thema	Schlüsselbegriffe
Windows-Betriebssystem	Windows ist die Betriebssystemfamilie, die mit Windows 1.0 begann. Windows 1.0 wurde am 20. November 1985 als Zusatzkomponente des MS-DOS-Betriebssystems veröffentlicht. Die Familie umfasst auch Mitglieder für eingebettete Geräte (Windows Embedded) und Mobilgeräte (Windows CE, Windows Mobile und Windows Phone). Die aktuellen Versionen sind Windows 8 und Windows Phone 8.
Windows-APIs	Anwendungen unter Windows können auf die zugrundeliegenden Dienste des Betriebssystems über die Windows Application Programming Interfaces (APIs) zugreifen.
Die Programmiersprachen C und C++	Als zwei Programmiersprachen, mit denen Windows-Anwendungen entwickelt werden können, nutzen C und C++ alle Funktionen des Betriebssystems, einschließlich Hardware-spezifischer und sehr systemnaher Funktionen. Diese Sprachen wurden seit den Anfangstagen von Windows für die Anwendungsentwicklung verwendet. C++ erweitert C um objektorientierte Funktionen und hat sich seit der Einführung vor über 25 Jahren zu einer ausgereiften Programmiersprache entwickelt.
Die Programmiersprache Visual Basic	Die Programmiersprache Visual Basic wurde 1991 veröffentlicht. Sie ist eine revolutionäre Programmiersprache, die den Programmieraufwand erheblich verringert, indem sie High-Level-UI-Konzepte wie etwa Forms, Steuerelemente und Ereignisse zur Verfügung stellt. Die Sprache bietet auch eine nahtlose Integration mit Component-Object-Model-(COM-)/Object-Linking-and-Embedding-(OLE-)Komponenten.
Microsoft .NET Framework	Das Microsoft .NET Framework ist eine Laufzeitumgebung, die zwischen dem Betriebssystem und den Anwendungen angesiedelt ist. Es soll hauptsächlich einen verwalteten Zugriff auf alle Dienste des zugrundeliegenden Betriebssystems ermöglichen, um Entwickler produktiver zu machen. C# und Visual-Basic-.NET sind die beiden am häufigsten verwendeten Sprachen, um Anwendungen für das Microsoft .NET Framework zu erstellen.
WPF und Silverlight	Windows Presentation Foundation (WPF) und Silverlight sind moderne und ausgefeilte UI-Plattformen, die auf dem Microsoft .NET Framework laufen. Sie können damit Anwendungen mit einer reichhaltigen UI mit einem flexiblen Layout, Multimediainhalten und Animationen erstellen. Beide Tools arbeiten mit fortgeschrittenen Techniken wie etwa Stilen, Vorlagen und Datenbindung, um die UI-Schicht von Anwendungen sehr effizient mit der zugrundeliegenden Logik zu verbinden.

Thema	Schlüsselbegriffe
Windows-8-Style-Anwendungen	Windows-8-Style-Anwendungen sind eine neue Form von Anwendungen, die in Windows 8 eingeführt wurden. Weil sie anders aussehen und Benutzerinteraktionen anders verwalten, unterscheiden sie sich grundsätzlich von traditionellen Desktopanwendungen. Windows-8-Style-Anwendungen beanspruchen den kompletten Bildschirm (das heißt, sie haben kein »Chrome«, keinen »Anwendungszierrat«), und innerhalb der Anwendung muss der Benutzer nicht mehr mehrere Fenster anzeigen und verwalten.