Mark Harrison, Michael McLennan

Effektiv Tcl/Tk programmieren

Deutsche Übersetzung von Birgit Krehl, Ralf Lübeck, Arnulf Mester, Dorothea Reder, Michael Sczittnick, Dirk Steinkamp

ADDISON-WESLEY

An imprint of Addison Wesley Longman, Inc.

Bonn • Reading, Massachusetts • Menlo Park, California
New York • Harlow, England • Don Mills, Ontario
Sydney • Mexico City • Madrid • Amsterdam

1 Tcl/Tk-Anwendungen entwerfen

Die meisten Menschen können die Grundlagen der Tcl/Tk-Programmierung in wenigen Stunden erfassen und bereits nach wenigen Tagen kleine lauffähige Programme schreiben. Ab diesem Zeitpunkt jedoch beginnt man sich verschiedene Fragen zu stellen, wie z.B.: Wie entwerfe ich größere Anwendungen? Wie entwerfe ich meine Bildschirmdarstellungen am besten? Wie kann ich meine Anwendung für eine Distribution vorbereiten?

In diesem Kapitel beschreiben wir den Entwurfsprozeß eines Layouts für eine kleine Anwendung von Anfang bis Ende. Wir werden ein einfaches Zeichenprogramm entwikkeln, wobei wir mit einer Handzeichnung unserer Idee beginnen werden. An einem ersten Prototypen können wir feststellen, ob uns das Look-and-Feel des Programms gefällt. Um unsere Implementierung zu beenden, werden wir schließlich noch einige Anbindungen und Prozeduren hinzufügen. Zudem zeigen wir Ihnen, wie Sie eine Entwicklung so planen können, daß der größte Teil des Programmcodes für Bibliotheken verwendet werden kann. Dadurch wird die Wiederverwendung von Programmcode unterstützt, was es Ihnen erleichtert, Anwendungen zu erstellen und zu warten.

1.1 Entwurfsprozeß von Anwendungen

Wie entwickelt man eine Anwendung mit Tcl und Tk? Die übliche Vorgehensweise kann wie folgt zusammengefaßt werden.

1. Überlegen Sie, wie die Anwendung aussehen soll. Entwerfen Sie ein paar Skizzen von den Haupt- und den Dialogfenstern. Es hilft dabei, ein paar Bücher über die Gestaltung von graphischen Oberflächen und von Benutzeroberflächen studiert zu haben. In Anhang B haben wir einige von uns bevorzugte Bücher aufgelistet.
2. Suchen Sie die Tk-Widgets aus, die bei der Realisierung der verschiedenen Skizzenelemente verwendet werden können. Gelegentlich wünscht man sich ein Widget, das nicht direkt von Tk zur Verfügung gestellt wird, zum Beispiel ein Notizbuch mit Registern oder eine Fortschrittsanzeige. Solche Dinge können aus Tk-Widgets

zusammengebaut werden. Die Zeichenfläche (engl. *canvas widget*) und das Text-Widget sind diesbezüglich äußerst nützlich (siehe Kapitel 4 und Kapitel 5 bzgl. weiterer Details). Eventuell finden Sie das, wonach Sie suchen, in Ihrer eigenen Codebibliothek oder im Tcl/Tk-Archiv im Internet (http://www.NeoSoft.com/tcl/).

3. Schreiben Sie den Tcl/Tk-Code, der die verschiedenen Widgets erzeugt und so zusammensetzt, daß es Ihren Skizzen ähnelt. Dies kann sehr schnell getan werden und erlaubt zudem, mit dem Erscheinungsbild der Fenster herumzuexperimentieren. In einigen Fällen möchte man von einigen Fenstern gewissermaßen eine »Light«-Version haben, die nicht alle Fähigkeiten aufweist. Man möchte zum Beispiel mit einem einfachen nackten Ausgabedialog beginnen, der noch nicht der vollständige Dialog mit allem Gebimmel und Getute ist. Wenn Sie allerdings einen produktreifen Dialog in Ihrer Bibliothek haben, können Sie diesen natürlich von Beginn an verwenden.

   Hinweis: Entwickeln Sie für jemand anderen, so ist dies der richtige Zeitpunkt, etwas Feedback bezüglich Ihres Entwurfs zu erhalten.

4. Stellen Sie fest, welche Komponenten oder Prozeduren sich dazu eignen, als Bibliotheksroutinen realisiert zu werden. Legen Sie wie in Abschnitt 8.2 genauer beschrieben die Infrastruktur der Bibliothek fest. Ein bißchen Planung an dieser Stelle kann eine Menge Zeit gegen Ende des Projektes sparen.

5. Fügen Sie Ihrem Programm Verhalten hinzu, indem Sie den Widgets Kommandos hinzufügen. Besitzt ein Widget keine Option, um das von Ihnen gewünschte Verhalten auszudrücken, so kann das Verhalten mit dem `bind`-Kommando hinzugefügt werden (siehe Kapitel 3 bzgl. weiterer Details).

   Ihr Programm kann andere Programme bemühen, um seine Aufgabe zu erfüllen. So kann zum Beispiel ein Zeichenprogramm das Druckprogramm des Systems – auf den meisten UNIX-Systemen also z.B. `lpr` – verwenden, um die Ausgaben zum Drucker zu senden (siehe Kapitel 7 bzgl. weiterer Details).

6. Geben Sie Ihrem Programm den letzten Schliff. Überprüfen Sie, ob Sie Widgetoptionen mit festen Einstellungen benutzen, die auch über variable Einstellungen in der Optionendatenbank festgelegt werden könnten; siehe hierzu auch Abschnitt 8.1.1. Fügen Sie Ihrer Anwendung mit Hilfe des Programmcodes aus Abschnitt 6.7.2 Sprechblasenhilfetexte hinzu. Benötigt Ihre Anwendung lange, bis sie gestartet ist, so verwenden Sie eine Ladeanzeige, z.B. die in Abschnitt 8.1.3, um dem Anwender anzuzeigen, daß das Programm arbeitet.

7. Testen Sie Ihr Programm bis zum Abwinken. Syntaxfehler im Tcl-Programmcode findet man erst, wenn dieser Programmcode auch ausgeführt wird.

8. Verpacken Sie Ihr Programm, damit es leicht distribuiert und installiert werden kann (weitere Details finden Sie in Kapitel 8). Verwenden Sie hierfür besondere Aufmerksamkeit, denn dies wird der erste Kontakt eines Kunden mit Ihrem Programm sein und sollte daher einen positiven Eindruck hinterlassen!

1.2 Eine kleine Anwendung

Wir werden nun schrittweise eine kleine Anwendung entwickeln – einen »Skizzenblock« (engl. *sketchpad*), auf dem man Bilder zeichnen und mit der Maus herumkritzeln kann. Wir beginnen mit einer einfachen Skizze des Gesamtentwurfs und verbessern sie allmählich zu einer gebrauchsfähigen Anwendung.

1.2.1 Entwurf der Anwendung

Zunächst überlegen wir uns, welche Fähigkeiten unser Skizzenblockprogramm besitzen soll. Da das Beispiel klein bleiben soll, konzentrieren wir uns auf ein paar wichtige Eigenschaften.

- Man soll durch Drücken der Maustaste und anschließendes Herumfahren zeichnen können.
- Man soll die Zeichenfläche löschen können.
- Man soll die Zeichenfarbe wählen können.
- Man soll die x- und y-Koordinaten des Mauszeigers sehen können.
- Man soll die Anwendung verlassen können.

Nun überlegen wir uns, wie wir diese Fähigkeiten steuern können. Wenn man eine neue Anwendung entwirft, sollte man sich andere Programme, die man benutzt, in Erinnerung rufen und das eigene Programm danach ausrichten. Zum Beispiel besitzen die meisten Programme oben eine Menüleiste mit einigen ausrollbaren Menüs. Auf diese Weise kann man es einem Anwender leichtmachen, die Eigenschaften des eigenen Programms herauszufinden. Bei genauer Beobachtung stellt man fest, daß die meisten Anwendungen Menüs mit den Bezeichnungen File, Edit und View bzw. (in einer deutschen Version) Datei, Bearbeiten und Ansicht/Darstellung besitzen. Das File-Menü besitzt Einträge zum Laden und Speichern sowie einen Beenden-Eintrag (Exit). Das Edit-Menü besitzt Einträge wie Cut und Paste, also zum Ausschneiden und Einfügen. Und das View-Menü enthält Einflußmöglichkeiten bezüglich der Darstellung, also Symbolleisten, Vergrößern/Verkleinern, eventuell ein Raster oder Lineale. Folgt man diesen Konventionen, werden Millionen von Anwendern kein Handbuch benötigen, da sie intuitiv die Anwendung zu nutzen wissen.

Abbildung 1.1 stellt unsere erste Vorstellung vom Aussehen unseres Skizzenblocks dar. Oben soll es eine Menüleiste mit zwei ausrollbaren Menüs geben. Das File-Menü enthält einen Exit-Eintrag, und das Edit-Menü besitzt einen Clear-Eintrag, um die Zeichenfläche zu löschen.

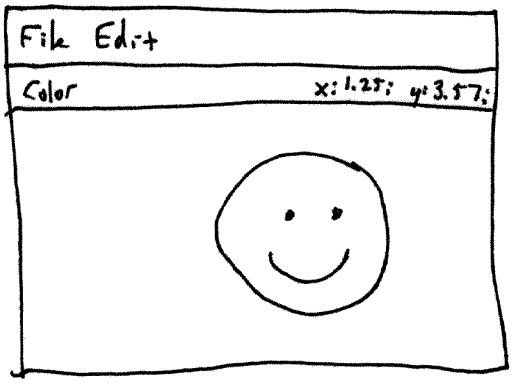


Abbildung 1.1: Skizzenblockprogramm, erste Idee.

Es ist keine gute Idee, neben den Menüs auch alles andere in der Menüleiste unterzubringen. Daher haben wir unter der Menüleiste eine weitere Zeile, die die Zeichenfarbe sowie die Position des Mauszeigers anzeigt. Darunter liegt die Zeichenfläche und belegt den größten Bereich des Bildschirms.

Es gibt etliche Möglichkeiten, die Zeichenfarbe festzulegen. Wenn man eine Funktion wie diese gestaltet, sollte man sich alle Möglichkeiten vor Augen führen und darüber abwägen. Zum Beispiel könnten wir ein Eingabefeld hinzufügen, in das man den Farbnamen eingibt. Viele Anwender würden über dauerndes Eintippen murren und unter Umständen den Farbnamen falsch schreiben. Wir könnten auch eine Liste mit allen Farbnamen anbieten, die allerdings eine Menge Platz im Hauptfenster einnehmen würde. Oder wir bieten ein Dialogfenster mit einem Farbzirkel an. Die meisten Menschen verstehen diese intuitive Auswahl einer Farbe. Wenn wir jedoch unsere Anwendung damit ausstatten und die Farbauswahl ausprobieren, werden wir feststellen, daß eine Änderung der Farbe nicht ganz mühelos ist. Man muß das Dialogfenster aufrufen, den Mauszeiger auf den Farbzirkel schieben und das Fenster wieder schließen. Wenn man nur ein paar einfache Farben wie Rot, Grün und Blau benötigt, ist der Farbzirkel ein bißchen zu aufwendig.

Für unsere Beispielanwendung verwenden wir z.B. ein Farbmenü mit den Farben Schwarz, Weiß, Rot, Grün und Blau. Damit haben wir eine kompakte, schnelle und einfache Auswahlmöglichkeit geschaffen. Zu Beginn soll das reichen. Falls nötig, können wir den Farbzirkel später noch hinzufügen.

1.2.2 Entwurf des Bildschirms

Zunächst müssen wir entscheiden, welche Tk-Widgets wir zum Erstellen der in Abbildung 1.1 dargestellten Steuerung gebrauchen können. Die Zeichenfläche können wir mit einem `canvas`-Widget realisieren. Dort, wo man auf die Zeichenfläche klickt, erzeugen wir ein kleines farbiges Quadrat. Mit einem `label`-Widget kann man die x- und y-Koordinaten des Zeichenstiftes auf der Zeichenfläche anzeigen.

Die Menüleiste ist ein `frame`-Widget mit darin eingepackten Menüschaltflächen. Jede Menüschaltfläche besitzt sein eigenes `menu`-Widget, das die Menüeinträge enthält.

Um, wie oben beschrieben, die Zeichenfarbe festzulegen, nutzen wir einen weiteren Menüknopf inklusive seinem zugehörigen Menü. Falls wir später noch einen Farbzirkel hinzufügen möchten, können wir diesen mit einem weiteren `canvas`-Widget realisieren. Wie, wird in Abschnitt 4.4 erläutert.

Nun ist es an der Zeit, das Layout des Bildschirms festzulegen. Abbildung 1.2 zeigt die grundlegende Vorgehensweise. Wir suchen nach Widgets, die miteinander gruppiert werden sollten, und fügen einen Rahmen (engl. *frame*) hinzu, der die Gruppierung vornimmt. Dadurch wird uns das Anordnen erleichtert, wie wir in Abschnitt 2.1.4 noch sehen werden. Zur Bildung der Menüleiste können wir einen Rahmen verwenden, der die Menüknöpfe gruppiert. Mit einem weiteren Rahmen gruppieren wir die Zeichenfarbe und die Position des Zeichenstifts. Alles zusammen packen wir in das Hauptfenster, zuoberst den Rahmen für die Menüleiste, darunter den Steuerungsrahmen des Zeichenstifts und ganz unten die Zeichenfläche.

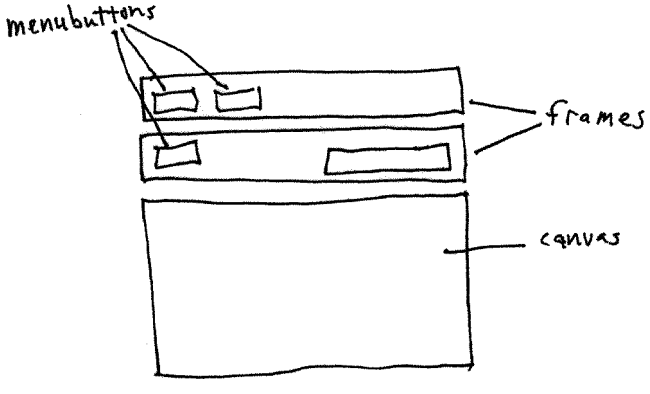


Abbildung 1.2: Layout des Bildschirms festlegen.

1.2.3 Ein Prototyp des Bildschirms

Nun werden wir ein paar rudimentäre Programmzeilen schreiben, damit wir unser Layout zu sehen bekommen. Der erste Teil geht uns leicht von der Hand. Wir erzeugen die Menüleiste `.mbar` und packen sie wie folgt in das Hauptfenster:

```
frame .mbar -borderwidth 1 -relief raised
pack .mbar -fill x
menubutton .mbar.file -text "File" -menu .mbar.file.m
pack .mbar.file -side left
menu .mbar.file.m
.mbar.file.m add command -label "Exit"

menubutton .mbar.edit -text "Edit" -menu .mbar.edit.m
pack .mbar.edit -side left
menu .mbar.edit.m
.mbar.edit.m add command -label "Clear"
```

Bis hierhin haben wir uns nur mit dem Aussehen des Programms beschäftigt – nicht damit, wie es funktioniert – daher haben wir für die Menüeinträge Angaben wie die Option `-command` weggelassen.

Wir erzeugen das Menü für die Zeichenfarbe und packen es wie folgt:

```
frame .style -borderwidth 1 -relief sunken
pack .style -fill x

menubutton .style.color -text "Color" -menu .style.color.m
pack .style.color -side left
menu .style.color.m
.style.color.m add command -label "Black"
.style.color.m add command -label "Blue"
.style.color.m add command -label "Red"
.style.color.m add command -label "Green"
.style.color.m add command -label "Yellow"
```

Wir erzeugen die Positionsangabe und packen diese wie folgt:

```
label .style.readout -text "x: 0.00 y: 0.00"
pack .style.readout -side right
```

Und wir erzeugen den Zeichenbereich und packen ihn wie folgt:

```
canvas .sketchpad -background white
pack .sketchpad
```

Abbildung 1.3 zeigt das Ergebnis, das zu diesem Zeitpunkt der Entwicklung ganz passabel wirkt. Entwickeln wir für jemanden anderen, sollten wir uns nun Bestätigung für unseren Entwurf besorgen. Der Kunde könnte Änderungswünsche haben, was wiederum Auswirkungen auf die Arbeitsweise der Anwendung haben könnte. Wir sollten

die Reaktion des Kunden kennen, bevor wir mit der weiteren Ausgestaltung zu viel Zeit verbringen.

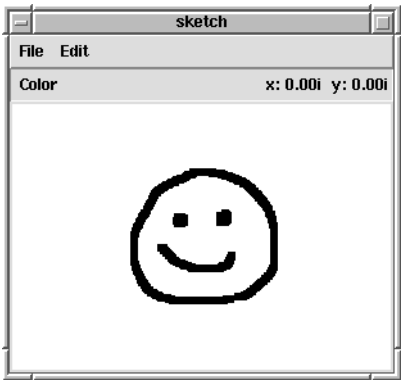


Abbildung 1.3: Skizzenblockprogramm, erstes Bild.

1.2.4 Bibliotheksanalyse

Bevor wir den Programmtext weiter fortführen, sollten wir nach Möglichkeiten Ausschau halten, die uns die Arbeit vereinfachen. Finden wir dieselbe Widgetkombination immer und immer wieder, können wir eine Prozedur zu ihrer Erzeugung schreiben.

Angenommen, wir planen zum Beispiel, eine Option anzubieten, um die Hintergrundfarbe bestimmen zu können. Hierfür benötigen wir ein weiteres Farbmenü. Wir könnten den Programmtext des Zeichenfarbmenüs duplizieren und ein paar Änderungen wie die folgenden durchführen:

```
menubutton .style.bg -text "Background" -menu .style.bg.m
pack .style.bg -side left

menu .style.bg.m
.style.bg.m add command -label "Black" -command {set bg black}
.style.bg.m add command -label "Blue" -command {set bg blue}
.style.bg.m add command -label "Red" -command {set bg red}
.style.bg.m add command -label "Green" -command {set bg green}
.style.bg.m add command -label "Yellow" -command {set bg yellow}
```

Doch statt dessen sollten wir eine Prozedur zur Erzeugung von Farbmenüs schreiben:

```
proc cmenu_create {win title cmd} {
    menubutton $win -text $title -menu $win.m

    menu $win.m
    $win.m add command -label "Black" -command "$cmd black"
    $win.m add command -label "Blue" -command "$cmd blue"
```

```
    $win.m add command -label "Red" -command "$cmd red"
    $win.m add command -label "Green" -command "$cmd green"
    $win.m add command -label "Yellow" -command "$cmd yellow"
}
```

Diese Prozedur erhält drei Parameter, nämlich die drei Dinge, die je Farbmenü anders sind: Einen Widgetbezeichner, einen Titel für den Menüknopf und den Befehl, der bei Auswahl einer Farbe ausgeführt werden soll. Die beiden Farbmenüs unserer Anwendung könnten wir dann wie folgt mit dieser Prozedur erzeugen:

```
cmenu_create .style.color "Color" {set color}
pack .style.color -side left

cmenu_create .style.bg "Background" {.sketchpad configure -bg}
pack .style.bg -side left
```

Zu beachten ist, daß die zu übergebenden Befehle zur Behandlung der Farbänderung nicht vollständig sind. Sie sind nur der erste Teil eines Befehls, den wir auch *Befehlspräfix* (engl. *command prefix*) nennen. Jeder Eintrag im Farbmenü hängt dem Befehlspräfix seinen Farbnamen am Ende an, so daß jeder Eintrag zur Farbfestlegung etwas anderes macht.

Offensichtlich sparen wir durch die Verwendung einer Prozedur jede Menge Programmtext und verringern zusätzlich die Gefahr, daß wir bei weiteren Farbmenüs, die wir der Anwendung hinzufügen, Fehler einbauen. Sollten wir uns dazu entschließen, das Farbmenü zu verbessern, müssen wir unsere Änderungen nur noch in einer Prozedur vornehmen.

Eine Komponente wie das Farbmenü könnte in vielen Anwendungen von Nutzen sein. Wir sollten etwas mehr Zeit aufwenden, um das Aussehen zu verbessern und es mit ein paar guten Prozeduren auszustatten. Dies geschieht in Abschnitt 8.2.3. Wären wir damit fertig, könnten wir ein Farbmenü etwa auf die folgende Art und Weise erzeugen:

```
colormenu_create .style.color
pack .style.color -side left
```

Da uns dieser Programmcode zur Verfügung steht, sollten wir ihn jetzt in unsere Skizzenblockanwendung einbauen. Die folgenden beiden Zeilen ersetzen den oben verwendeten Menüprogrammtext. Wenn wir auf der Zeichenfläche Markierungen hinzufügen, befragen wir das Farbmenü nach der aktuellen Farbe zum Beispiel wie folgt:

```
set color [colormenu_get .style.color]
```

Integrieren wir die Farbmenüprozedur in eine Tcl/Tk-Bibliothek, können wir sie in künftigen Projekten wiederverwenden. In diesem Buch werden wir etliche Komponenten entwickeln. Alle folgen den im Abschnitt 8.2 beschriebenen Entwurfsmustern.

1.2.5 Hinzufügen von Verhalten

Nun fügen wir unserer skelettartigen Anwendung des Prototyps Programmtext hinzu, der das Verhalten beschreibt. Eine der einfachsten Anpassungen ist, das File-Menü dahingehend zu ändern, daß man die Anwendung verlassen kann. Dazu muß man den Befehl `exit` dem Exit-Eintrag zum Beispiel wie folgt hinzufügen:

```
.mbar.file.m add command -label "Exit" -command exit
```

Als nächstes können wir das Edit-Menü anpassen, um den Clear-Eintrag zu realisieren. Wie wir in Kapitel 4 sehen werden, kann man die Zeichenfläche löschen, indem man ihr mitteilt, daß sie all ihre Elemente entfernen soll:

```
.mbar.edit.m add command -label "Clear" -command {
    .sketchpad delete all}
```

Das trickreichste Stück Programmtext ist jedoch für das Skizzieren notwendig. Die Zeichenfläche (engl. *canvas*) unterstützt von sich aus kein Skizzieren. Wir müssen mittels des Befehls `bind` neues Verhalten hinzufügen. In Kapitel 3 erläutern wir Anbindungen im Detail, nur soweit vorweg die grundlegende Idee: Immer wenn etwas mit der Zeichenfläche oder mit irgendeinem anderen Widget passiert, empfängt dieses ein Ereignis (engl. *event*), welches beschreibt, was passiert ist. Wenn man zum Beispiel den linken Mausknopf drückt, empfängt es das Ereignis `<ButtonPress-1>`. Per Voreinstellung ignoriert die Zeichenfläche diese Ereignisse. Jedoch können wir mit dem Befehl `bind` ein Skript registrieren, welches das Ereignis behandeln soll.

Betrachten wir zunächst die Darstellung der Position des Mauszeigers, während sich dieser über die Zeichenfläche bewegt. Dazu binden wir das Ereignis `<Motion>` (dt. Bewegung) wie folgt an:

```
bind .sketchpad <Motion> {sketch_coords %x %y}
```

Die Bewegung des Mauszeigers über den Teil des Bildschirms, den `.sketchpad` belegt, erzeugt eine Reihe von `<Motion>`-Ereignissen. Jedes Ereignis repräsentiert die in gewissen Zeitabständen ermittelte Position des Mauszeigers. Wird die Maus langsam bewegt, wird eventuell für jeden Bildpunkt auf ihrem Weg ein Ereignis erzeugt. Doch normalerweise sind es sich etwas stärker unterscheidende Koordinatenwerte.

Mittels der `<Motion>`-Anbindung ersetzt Tk die Angaben `%x` und `%y` jedes Ereignisses durch die *x*- und *y*-Koordinate und ruft dann die Prozedur `sketch_coords` auf. Letztere sieht in etwa wie folgt aus:

```
proc sketch_coords {x y} {
    set size [winfo fpixels .sketchpad 1i]
    set x [expr $x/$size]
    set y [expr $y/$size]
    .style.readout configure \
        -text [format "x: %6.2fi  y: %6.2fi" $x $y]
}
```

Um die neuen Koordinaten darzustellen, ändern wir den Text der Beschriftung `.style.readout`. Wir könnten die Koordinaten direkt mitteilen, jedoch ist für den Anwender eine zum Beispiel in Zoll umgewandelte Angabe nützlicher. Wir verwenden den Befehl `winfo fpixels`, um die Anzahl Bildpunkte (engl. *pixel*) je Zoll auf dem Skizzenblock herauszufinden, und passen die *x*- und *y*-Koordinate entsprechend an. Wir geben nur zwei Ziffern hinter dem Dezimalpunkt aus – schließlich ist der Anwender kein Computer! Daher verwenden wir den Befehl `format`, um eine hübsche Zeichenkette wie `x: 1.49i  y: 2.51i` zu erzeugen. Diese wird dann angezeigt.

Nun müssen wir das Zeichnen behandeln. Wir täuschen die Illusion mit einem Stift zu zeichnen vor, indem wir während der Mausbewegung der Zeichenfläche quadratische Kästchen hinzufügen. Dafür müssen wir zwei Ereignisse anbinden: das Ereignis `<ButtonPress-1>`, so daß durch das Drücken des Mausknopfes gewissermaßen »Tinte« auf die Zeichenfläche tropft; und das Ereignis `<B1-Motion>`, so daß der Stift sozusagen »zeichnet«, während sich die Maus bei gedrücktem Mausknopf bewegt:

```
bind .sketchpad <ButtonPress-1> {sketch_box_add %x %y}
bind .sketchpad <B1-Motion>     {sketch_box_add %x %y}
```

Tk ersetzt `%x` und `%y` mit der Position des Mauszeigers zur Zeit der Ereignisauslösung und ruft dann die wie folgt definierte Prozedur `sketch_box_add` auf:

```
proc sketch_box_add {x y} {
    set x0 [expr $x-3]
    set x1 [expr $x+3]
    set y0 [expr $y-3]
    set y1 [expr $y+3]
    set color [colormenu_get .style.color]

    .sketchpad create rectangle $x0 $y0 $x1 $y1 \
        -outline "" -fill $color
}
```

Wir wollen ein Quadrat mit 6 mal 6 Bildpunkten Größe zeichnen, dessen Mittelpunkt bei der Koordinate (*x*,*y*) liegt. Daher subtrahieren wir von *x* und *y* je 3 Bildpunkte, um die obere linke Ecke, und addieren jeweils 3 Bildpunkte, um die untere rechte Ecke des Quadrats zu berechnen. Die Zeichenfarbe erhalten wir durch Aufruf von `colormenu_get` mit dem Namen des Farbmenüs. Wie das genau funktioniert ist in Abschnitt 8.2.3 beschrieben, doch für jetzt nehmen wir an, daß sie einen Farbnamen wie Schwarz, Rot oder Grün liefert. Schließlich tropfen wir ein kleines quadratisches Stück Tinte auf die Zeichenfläche, indem wir ein Rechteck erzeugen. Umsichtig setzen wir die Außenrandfarbe auf die leere Zeichenkette, um die Voreinstellung schwarz zu überschreiben.

1.2.6 Feinschliff durchführen

Da nun das Skizzenprogramm in seiner Grundfassung läuft, können wir es jetzt ein wenig aufpolieren. Die folgenden Gedanken sollte man beim Vervollständigen seiner Anwendungen berücksichtigen.

Zunächst sollte man sich alle Fenster seiner Anwendung sorgfältig ansehen.

- Sind die Widgets alle zusammengequetscht? Sind irgendwelche Widgets an den Fensterrand gedrängt? Ist dies der Fall, so sollte man, wie in Abschnitt 2.1.2 beschrieben, ein paar Abstände bei den Befehlen `pack` und `grid` mittels der Optionen `-padx` und `-pady` hinzufügen.
- Besitzt jedes Fenster oben einen passenden Titel? Die Titelzeile eines Fensters kann man mit dem Befehl `wm title` ändern (siehe Abschnitt 6.3). Die Titelzeile für unser Skizzenblockprogramm können wir zum Beispiel wie folgt festlegen:

```
wm title . "sketch"
```

Als nächstes sollte man sich alle statisch kodierten Widgetoptionen im Programm ansehen. Konstante Angaben für die Option `-text` einer Beschriftung oder für die Option `-command` eines Knopfes sind völlig in Ordnung. Bei statischen Farb- und Zeichensatzangaben sollte man sich allerdings überlegen, ob man diese nicht in die Optionendatenbank mit aufnimmt. Wird zum Beispiel eine Schrift fest angegeben, so kann ein Anwender, dem diese Schrift nicht zur Verfügung steht, das Programm nicht benutzen.

Im Skizzenblockprogramm zum Beispiel haben wir die Hintergrundfarbe der Zeichenfläche wie folgt statisch kodiert festgelegt:

```
canvas .sketchpad -background white
```

Doch was ist, wenn der Anwender einen Hintergrund in Schwarz oder antikem Weiß haben will? Daher ist es besser, wenn die Zeichenfläche wie folgt erzeugt wird:

```
canvas .sketchpad
```

und der Optionendatenbank eine Ressource wie folgt hinzugefügt wird:

```
option add *sketchpad.background white startupFile
```

Hierdurch wird die Hintergrundressource für jedes Widget mit dem Namen `sketchpad` auf Weiß (engl. *white*) gesetzt. In dieser Anwendung gibt es nur einen Skizzenblock (`sketchpad`), jedoch könnten wir diesen verallgemeinern und mehrere Skizzenblöcke unterstützen, die dann alle weiß wären. In Abschnitt 8.1.1 werden wir mehr über die Optionendatenbank erfahren, insbesondere, wie sie funktioniert.

Als nächstes sollte man Tastaturbefehle für die Menüs definieren. Für einen Anfänger ist das Umherstöbern in den Menüs ganz nützlich, doch erfahrene Anwender finden es

nur lästig, immer wieder die gleichen Menüs zu öffnen. Daher ist es sinnvoll, für häufig verwendete Funktionen wie Ausschneiden und Einfügen die sogenannten Shortcuts, also kurze Tastaturbefehle, anzubieten.

Und letztendlich sollte dem Programm noch eine Onlinehilfe hinzugefügt werden. Dazu fügt man dem Programm ein Help-Menü mit einer Themenliste und ein Text-Widget zur Darstellung des Hilfetextes hinzu. In Abschnitt 5.3 wird dies demonstriert.

Oder man verwendet die in Abschnitt 6.7.2 beschriebene Sprechblasenhilfe (engl. *balloon help*). Dem Skizzenblockprogramm können wir zum Beispiel mit den folgenden einfachen Befehlen eine solche Hilfefunktion hinzufügen:

```
balloonhelp_for .style.color {Pen Color:
Selects the drawing color for the canvas}
balloonhelp_for .style.readout {Pen Location:
Shows the location of the pointer on the drawing canvas (in inches)}
balloonhelp_for .sketchpad {Drawing Canvas:
Click and drag with the left mouse button to draw in this area}
```

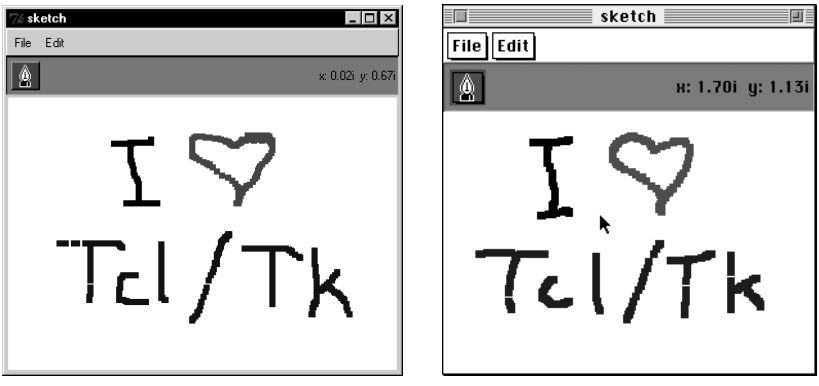
Verharrt der Mauszeiger auf dem Farbmenü, der Positionsanzeige oder der Zeichenfläche eine Weile, erscheint ein Fenster mit einem der hier angegebenen kurzen Erläuterungen. Wie man sieht, ist das Hinzufügen einer Sprechblasenhilfe nicht gerade schwierig und es erleichtert einem neuen Anwender, das Programm zu verwenden.

Zudem kann man dem File-Menü einen Eintrag namens About... hinzufügen. Dadurch kann zum Beispiel ein Dialogfenster mit dem Namen des Autors, einem Copyright und einer Telefonnummer der technischen Abteilung angezeigt werden.

Mit all diesen Verbesserungen wirkt unser einfaches Skizzenprogramm wie eine voll ausgereifte Anwendung. Das Endprodukt ist in Abbildung 1.4 abgebildet.

Abbildung 1.4: Endgültige Version der Skizzenblockanwendung, mit Sprechblasenhilfe.

Dasselbe Skript läuft plattformübergreifend auf UNIX-, Windows 95/NT- und auf Macintosh-Systemen. Unter Windows sieht es wie eine Windows-Anwendung aus; auf einem Mac sieht es, wie in Abbildung 1.5 dargestellt, wie eine Mac-Anwendung aus. In Kapitel 9 werden Portabilität diskutiert und einige Richtlinien vorgestellt, die es erleichtern, Skripts zu entwickeln, die für alle drei Plattformen portabel sind.



(a) (b)

Abbildung 1.5: Die Skizzenblock-Anwendung läuft plattformübergreifend auf (a) Windows 95/NT- und (b) Macintosh-Systemen.

1.2.7 Testen des Programms

Bevor man ein Programm vertreibt, sollte man es gründlich testen. In C oder C++ geschriebene Programme werden von einem Übersetzer auf viele Fehler hin durchforstet. Bei Tcl jedoch stellt man Fehler erst zur Laufzeit fest. Da alles interpretiert wird, muß man das Programm ausführen, um Fehler zu entdecken. Alle Prozeduren, Schleifen und bedingten Verzweigungen müssen ausgeführt werden. Sucht man keine Fehler, findet der Kunde diese unter Umständen und ist eventuell die längste Zeit ein Kunde gewesen!

Beim Testen sollte man folgende Probleme berücksichtigen.

- Es sollte versucht werden, jedes Fenster in seiner Größe zu verändern. Dadurch werden Fehler beim Packen und beim Rasterlayout aufgedeckt (siehe Kapitel 2.1.5–2.1.6 und Abschnitt 2.2.3). Betrachten wir als Beispiel unser Skizzenblockprogramm. Verkleinert man das Fenster wie in Abbildung 1.6(a) dargestellt, so sieht alles okay aus. Vergrößert man jedoch das Fenster, so entdecken wir einen Packfehler. Statt daß die Zeichenfläche größer wird, behält sie ihre Größe bei, und es ensteht nur toter Raum drumherum. Dies kann man korrigieren, indem man beim Packen der Zeichenfläche Expandieren (engl. *expand*) und Füllen (engl. *fill*) mit angibt.

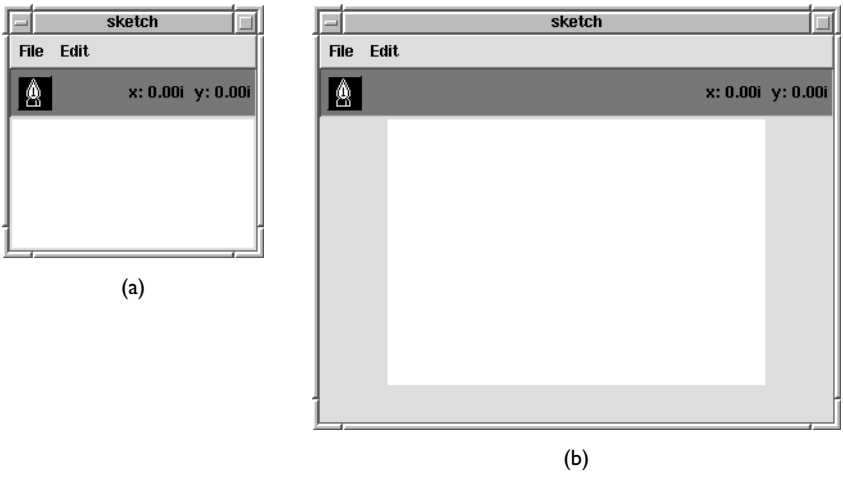


Abbildung 1.6: Skizzenblockanwendung mit dem Hauptfenster (a) verkleinert und (b) vergrößert. Dadurch wird ein Packproblem sichtbar.

Ein Programm sollte man sich bei verschiedenen Bildschirmauflösungen ansehen. Wird es bei einer Bildschirmauflösung von 640×480 gut dargestellt?

Besitzt ein Fenster eine minimale Größe oder möchte man eine Änderung der Größe verbieten, kann man dies dem Fenstermanager mitteilen (siehe Abschnitt 6.3.2).

- Man sollte die Tastaturanbindungen überprüfen. Wandert der Fokusrahmen wie gewünscht weiter, wenn man die Tab-Taste drückt? Wenn nicht, sollte man die Reihenfolge der `pack`- und `grid`-Befehle dahingehend ändern, daß sie die gewünschte Tabreihenfolge wiedergibt.

  Funktioniert etwas nicht, wenn eine der Tasten CapsLock oder NumLock aktiviert ist? Insbesondere sollten die Tastaturkürzel der Menüs überprüft werden.

- Was passiert, wenn man ein Dialogfenster nicht über Knöpfe wie OK, Cancel oder Beenden verläßt, sondern die Optionen Quit oder Close des Windowmanagers verwendet? Läßt sich das Fenster wieder aktivieren? Wenn nicht, sollte man wie in Abschnitt 6.5 beschrieben das Protokoll `WM_DELETE_WINDOW` berücksichtigen.

- Benutzt die Anwendung Dateien, sollte man versuchen, Dateien zu lesen, die nicht existieren. Ebenso sollte man versuchen, in schreibgeschützte Dateien zu schreiben oder in eine Datei zu schreiben, wenn die Diskette bzw. Festplatte voll ist. Erhält man den Fehlerdialog von Tcl, sollten `catch`-Befehle an geeigneter Stelle hinzugefügt werden, um eine bessere Fehlerbehandlung zu haben (siehe Abschnitt 6.4).

- Die Anwendung sollte man auf Windows- und auf Macintoshplattformen laufen lassen, um zu sehen, ob sie auch weiterhin gut aussieht.

- Dieser Test hilft zudem festzustellen, ob man Annahmen über das Betriebssystem getroffen hat. Verläßt man sich auf UNIX-Programme, wie `rm` oder `lpr` zum Beispiel, wird man es auf anderen Plattformen sofort bemerken.

1.2.8 Ein Programm verpacken

Nun ist das Programm fertiggestellt, getestet und bereit, versandt zu werden. Doch in welcher Form liefert man es den Kunden aus? Es gibt mehrere Möglichkeiten.

- Die Kunden haben die neueste Version einer `wish` auf ihren Maschinen installiert. Dieses Programm wurde in Verbindung mit dem Betriebssystem geliefert, oder die Kunden nutzen es für eigene Zwecke. In solch einem Fall muß man nur die Skriptdateien der Anwendung weitergeben.
- Die Kunden besitzen keine `wish` und wollen auch nichts davon wissen. In diesem Fall muß man eine `wish` für die entsprechende Plattform des Kunden erzeugen und eine vollständige Distribution davon zusammen mit den Skriptdateien versenden. Die Kunden können dann die Distribution auf ihren Maschinen auspacken und ein Installationsprogramm, üblicherweise »install« genannt, ausführen, welches die notwendigen Dateien und Programme an die entsprechenden Stellen plaziert. In Abschnitt 8.3 zeigen wir, wie man ein `wish`-Skript zur Installation schreibt. Schließlich versendet man eine `wish` und kann sie auch hierfür ausnutzen.
- Ihre Kunden sind gelegentliche Anwender mit Zugang zum Internet. In solch einem Fall kann man die Anwendung so verpacken, daß sie innerhalb eines Webbrowsers läuft. Dafür muß man eine Webseite mit einem Verweis auf die Anwendung erzeugen und das Anwendungsskript auf dem Webserver speichern. In Abschnitt 8.4 wird gezeigt, wie das geht. Sieht sich ein Kunde die Webseite an, lädt der Webbrowser automatisch das Anwendungsskript und führt es anschließend aus. Browserprogramme wie Netscape Navigator besitzen sogenannte Plug-in-Module, die dies bewerkstelligen. Solange ein Kunde das entsprechende Plug-in-Modul auf seinem Rechner installiert hat, kann er beliebige Tcl/Tk-Anwendungen ausführen.

  Die Anwendung wird in der Webseite, die der Kunde sich ansieht, integriert sein, anstatt in einem eigenen Fenster abzulaufen. Doch abgesehen davon, sieht sie wie eine normale Tcl/Tk-Anwendung aus und läuft auf der Maschine des Kunden ab, so daß sie durch größeren Netzwerkverkehr nicht verlangsamt werden kann. Dieses Verfahren läßt sich für kleine Anwendungen (die somit schnell geladen werden können) gut einsetzen und ist auch ideal, wenn man viele Anwender erreichen möchte, die die Anwendung nur gelegentlich nutzen wollen.

Bei unserem einfachen Skizzenblockprogramm gehen wir davon aus, daß unsere Kunden mit ihrer neuen Tcl/Tk-Anwendung herumexperimentieren werden. Deshalb werden wir nur das Skript versenden. Die endgültige Version des Skizzenblocks

verwendet allerdings auch ein paar Bibliotheksfunktionen für das Farbmenü, die Sprechblasenhilfe sowie eine spezielle Bitmap für die Zeichenstiftfarbe. Dies ist für viele Anwendung nichts Ungewöhnliches. Meistens wird ein Kunde etliche verschiedene Dateien benötigen, um eine Anwendung ablaufen lassen zu können. In Abschnitt 8.3 wird gezeigt, wie man eine vollständige Distribution erstellt und wie man sie installiert.

Da wir nun wissen, wie der grundlegende Entwurfsprozeß abläuft, wollen wir die Details von Tk etwas beleuchten. Wir werden dabei versuchen, unsere Ideen als Bibliotheksfunktionen zu realisieren, die Sie in Ihrer eigenen Anwendung verwenden können. Wenn Sie den Programmcode dafür einsetzen wollen, müssen Sie ihn nur von der auf Seite XV genannten Webseite herunterladen.