

# Programmierkonzepte in der Physikalischen Chemie

## 7. Grafische Nutzerschnittstellen (GUIs)

Albert-Ludwigs-Universität Freiburg



**UNI  
FREIBURG**

**Dr. Till Biskup**

Institut für Physikalische Chemie  
Albert-Ludwigs-Universität Freiburg  
Wintersemester 2013/14

## Besonderheiten von GUIs

### Bausteine von GUIs

- Übersicht über grundlegende Bausteine
- Eigenschaften

### Allgemeines zur GUI-Entwicklung

- Grundkonzepte und „Best Practices“
- Designentscheidungen
- Allgemeiner Entwicklungszyklus einer GUI

### GUI-Entwicklung in Matlab

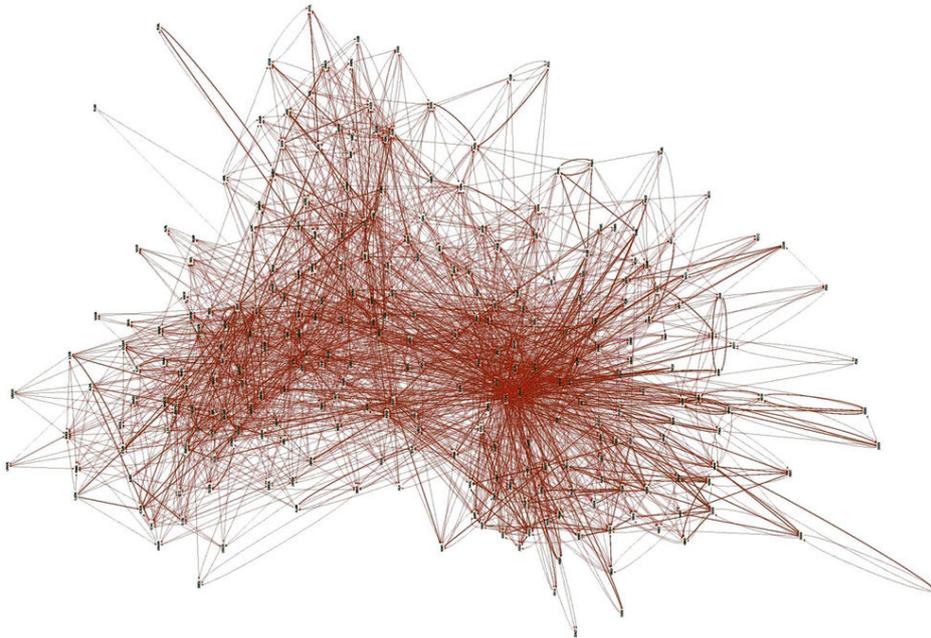
- Grundsätzliches Herangehen
- Besonderheiten von Matlab
- Hilfreiche Konzepte

# Besonderheiten von GUIs

Nicht linear und nicht vorhersagbar



UNI  
FREIBURG



Brain connectivity map of *C. elegans*



## Textbasierte Nutzerschnittstelle (CLI)

- ▶ Menüs und Nutzereingaben in einer Textkonsole
- ▶ Vollständig deterministisch (bis auf Nutzereingaben)
- ▶ Linear: immer nur eine Entscheidungsmöglichkeit
- ▶ Strukturiert, aber mit wenig Freiheiten

## Grafische Nutzerschnittstelle (GUI)

- ▶ Grafische Anordnung von Bedienelementen
- ▶ Reihenfolge der Ereignisse unvorhersehbar
- ▶ Nicht linear: beliebige Entscheidungsmöglichkeiten
- ▶ Große Freiheit: Alles (implementierte) jederzeit möglich.



### Besonderheiten von GUIs mit großer Auswirkung

- ▶ Nicht linear und nicht vorhersehbar
    - Nutzer kann jederzeit jedes implementierte Ereignis in beliebiger Reihenfolge auslösen.
  - ▶ Ereignisgetrieben
    - Grundsätzlich anderes Programmier-Paradigma
  - ▶ Parallel
    - Zeitlich parallele Verarbeitung mehrerer Prozesse
  - ▶ Keine Nachvollziehbarkeit der Bearbeitungshistorie
    - Muss per Hand implementiert werden
- ☞ Direkte, tiefgreifende Auswirkung auf die Programmierung



### Nicht linear und nicht vorhersehbar

- ▶ Reihenfolge der Aufgaben (Funktionsaufrufe) beliebig
  - Größtmögliche Freiheit des Nutzers
  - Reihenfolge der Datenverarbeitungsschritte nicht festgelegt
  - Freiheit bedingt immer auch Verantwortung
  - Nutzerführung durch intuitive Anordnung der Bedienelemente
  
- ▶ Zustand beim Funktionsaufruf erst zur Laufzeit bekannt, nicht bereits bei der Implementierung
  - Strikte Überprüfung aller Variablen und Objekte, auf die zugegriffen werden soll (Existenz, Zustand)
  - Kapselung der Routinen
  
- ▶ Paralleler Zugriff auf gemeinsame Ressourcen



## Ereignisgetriebene Programmierung

Der Programmfluss wird durch Ereignisse (*events*) gesteuert, das Programm *nicht* linear durchlaufen.

- ▶ Häufig zur GUI-Programmierung eingesetzt
- ▶ Zentrale Elemente
  - Ereignisse (*events*)
  - Hauptschleife, die auf Ereignisse „hört“ (*listener*)
  - Ereignisbehandlungsroutinen (*event handler, callbacks*)
- ▶ Parallele Programmierung
  - Mehrere Prozesse laufen parallel.
  - Kann zu unerwünschten Randeffekten führen



## Zeitlich parallel laufende Prozesse (Threads)

- ▶ Situation in GUIs
  - Auslösung eines Ereignisses ist (relativ) schnell
  - Abarbeitung der ausgelösten Aufgaben kann lange dauern
  - Nutzer erwarten eine schnelle Antwortzeit.
  
- ▶ Ereignisgetriebene Programmierung
  - Hauptschleife nimmt Ereignisse entgegen und ruft die Ereignisbehandlungsroutinen auf
  
- ▶ Problem
  - Ereignisbehandlung läuft (teilweise) parallel
  - Gleichzeitiger Zugriff auf gemeinsame Ressourcen
  - Ressourcen: Daten, GUI-Zustand, ...



## Threadsicherheit

Eine Komponente kann gleichzeitig von verschiedenen Programmbereichen mehrfach ausgeführt werden, ohne sich gegenseitig zu behindern.

- ▶ Gleichzeitige Manipulation von Daten durch mehrere Threads kann zu inkonsistenten Daten führen.
- ▶ Strategien zur Vermeidung von Inkonsistenzen
  - Semaphore
  - Mutex (*mutual exclusive*, gegenseitiger Ausschluss)
  - Synchronisierung
- ▶ GUI-Elemente sind i.d.R. *nicht* threadsicher.



## Wettlaufsituation (*race condition*)

Das Ergebnis einer Operation hängt vom zeitlichen Verhalten bestimmter Einzeloperationen ab.

- ▶ „Worst case“
  - Unvorhersehbares Verhalten
  - Schwer zu reproduzieren und zu beheben (Debugging verändert Verhalten)
  
- ▶ Konsequenzen für die GUI-Programmierung
  - Bewusstsein für die Probleme paralleler Programmierung
  - Kapselung einzelner Operationen
  - Strategien zur Zugriffssteuerung und Synchronisation



### Keine Nachvollziehbarkeit der Bearbeitungshistorie

- ▶ Konsequenz des Programmierparadigmas einer GUI
  - Ereignisgetrieben
  - Nicht linear und nicht vorhersehbar
  
- ▶ Problem für die (Natur-)Wissenschaften
  - Vollständige Nachvollziehbarkeit aller Details der Datenverarbeitung von entscheidender Bedeutung.
  - Reproduzierbarkeit der Datenverarbeitung (ggf. auch mit einem anderen Datensatz)
  
- ☞ Manuelle Implementierung einer Bearbeitungshistorie
  
- ☞ Konzept der Historie im Datensatz



### Historie – Dokumentation aller Verarbeitungsschritte

- ▶ Zielstellung
  - Nachvollziehbarkeit
  - Reproduzierbarkeit
  
- ▶ Felder für einen Eintrag
  - Name des Durchführenden
  - Datum
  - Name und Version der verarbeitenden Routine
  - Version des zugrundeliegenden Programms (z.B. Matlab)
  - Name und Version des Betriebssystems
  - ggf. sämtliche Eingabeparameter für die Routine
  
- 👉 Gewährleistet zusammen mit einer Versionsverwaltung (für den Code) die vollständige Reproduzierbarkeit.



### Textbasierte Nutzerschnittstellen (CLI)

- ▶ Relativ einfach zu implementieren.
- ▶ Lineare (festgelegte) Nutzerführung
- ▶ Notwendigkeit, alles textlich zu beschreiben

### Grafische Nutzerschnittstellen (GUI)

- ▶ Oft schnellerer Zugang für den Gelegenheitsnutzer
  - ▶ Wesentlich aufwendiger in der Programmierung
  - ▶ Matlab: Festlegung auf ein kommerzielles Programm
  - ▶ Plattformunabhängigkeit schwer zu gewährleisten
- ☞ Klare Abwägung der jeweiligen Kosten und Nutzen

# Bausteine von GUIs

Einfache Grundelemente – vielfältige Kombinationsmöglichkeiten



UNI  
FREIBURG





### Steuerelement (*widget, control*)

Interaktionselement in einer grafischen Benutzeroberfläche

- ▶ *widget* = *window* + *gadget*
- ▶ Steuerelemente im Programm mit Funktionen verknüpft
  - Werden ausgeführt, wenn das Element aktiviert wird
- ▶ Einfache Steuerelemente
  - Bsp.: Schaltfläche, Auswahlliste, ...
- ▶ Komplexe Elemente (vorgefertigt)
  - Bestehen aus einfachen Steuerelementen
  - Bsp.: Dialog zum Speichern einer Datei



### Grundlegende Bausteine

- ▶ Symbol (*Icon*)
  - grafisches Symbol
- ▶ Bezeichnungsfeld (*Label*)
  - Einfacher Text für Beschreibungen
  - Nicht-interaktiv
- ▶ Textfeld
  - Ein- oder mehrzeiliges Feld
  - Für die Ein- oder Ausgabe von Zahlen oder Text
- ▶ Schaltfläche (*Button*)
  - Fläche zum Anklicken mit der Maus
- ▶ Schieberegler (*Slider*)
  - Zur grafischen Auswahl von Werten



### Grundlegende Bausteine

- ▶ Auswahlkästchen (*Checkbox*)
  - Zwei Zustände: gesetzt/nicht gesetzt
- ▶ Optionsfeld (*Radiobutton*)
  - Auswahl (genau) einer Möglichkeit aus einer Gruppe
- ▶ Listenfeld (*Listbox*)
  - mehrzeilig, zur Auswahl
- ▶ Kombinationsfeld (*Combobox*)
  - Kombination aus Textfeld und Listenfeld
- ▶ Dropdown-Listenfeld (*Dropdown-Liste*)
  - Kombination aus schreibgeschütztem Textfeld und ausklappbarer Auswahlliste
  - Textfeld kann nur einen der Einträge der Liste annehmen



### Fensterspezifische Bausteine

- ▶ Titel
  - Prägnante Bezeichnung des aktuellen Fensters
- ▶ Menü
  - Auswahlliste mit Drop-Down-Listen
  - Grundlegende Funktionen eines Programms
- ▶ Symbolleiste
  - Mehrere Schaltflächen (Icons) für Programmfunktionen
- ▶ Bildlaufleiste (*Scrollbar*)
  - Verschiebt Fensterinhalt vertikal oder horizontal
- ▶ Statusleiste
  - Bereich am unteren Rand eines Fensters
  - Anzeige von Programmstatus und/oder Hilfetexten



### Bausteine zur Gliederung

- ▶ Baum
  - Hierarchisch gegliederte Auswahlliste
  - Aufklappbare Knoten
- ▶ Registerkarte (Reiter, *Tab*)
  - Mehrseitige Dialogfenster
  - Mehrere Dokumente pro Fenster
- ▶ Gruppenfeld (*Panel*)
  - Rahmen mit Überschrift
  - Gruppierung zusammengehörender Steuerelemente

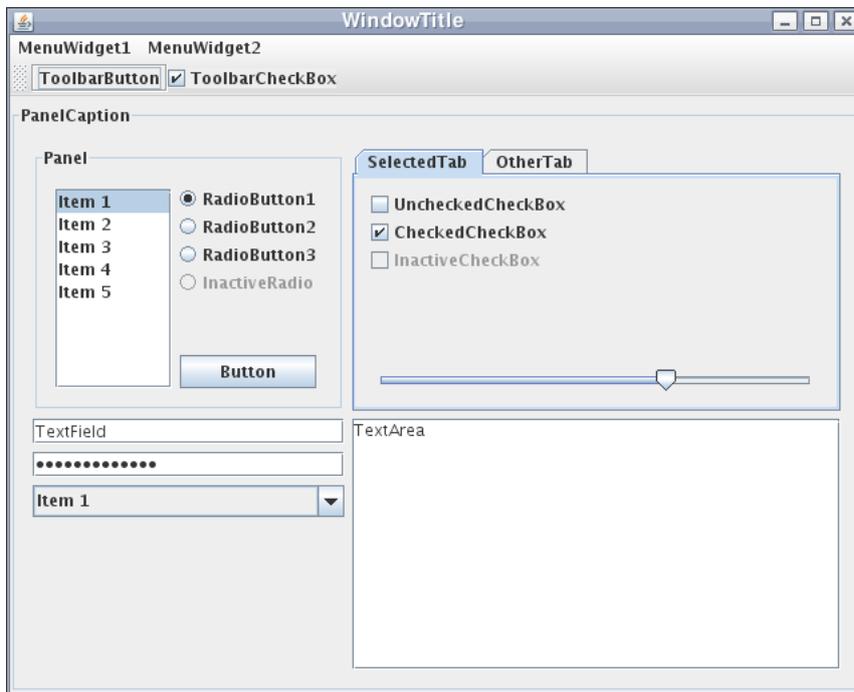


### Spezielle Bausteine

- ▶ Fortschrittsbalken
  - Anzeige des Fortschritts einer Operation
  - Rückmeldung an den Nutzer, dass etwas passiert
- ▶ Tabelle
  - Komplexes Element für Daten/Werte
  - Verhalten ähnlich einem Tabellenkalkulationsprogramm
- ▶ Achsen
  - Insbesondere bei der Datenverarbeitung
  - Mehrdimensionale Auftragung (numerischer) Daten
- ▶ Mauszeiger
  - Unterschiedliches Aussehen je nach Funktion
  - Beispiele: Hand, Sanduhr, Fadenkreuz, ...

# Bausteine von GUIs

## Übersicht über grundlegende Bausteine





### Bausteine können Ereignisse auslösen

- ▶ Ereignis (*event*)
  - Meist vom Nutzer ausgelöst (z.B. Mausklick)
  - Art des Ereignisses
  - Quelle des Ereignisses
  
- ▶ Hauptschleife (*listener*)
  - Hört auf die Ereignisse
  - Ruft die für ein Ereignis zuständigen Routinen auf
  - Jedes interaktive Element „registriert“ sich vorher.
  
- ▶ Ereignisbehandelnde Routine (*callback*)
  - Programmlogik, nicht eigentliche Datenverarbeitung
  - Sorgt u.a. für die Aktualisierung der GUI-Elemente



### Bausteine haben vier mögliche Zustände

#### 1 Aktiv

- Sichtbar
- Nutzerinteraktion löst Ereignis aus

#### 2 Inaktiv

- Sichtbar
- Keine Reaktion auf Nutzerinteraktion

#### 3 Sichtbar

- Aktiv oder inaktiv

#### 4 Unsichtbar

- Für den Nutzer unerreichbar
- Keine Nutzerinteraktion möglich



### Best Practice

Bewährte, optimale bzw. vorbildliche Methoden, Praktiken oder Vorgehensweisen; unverbindliche Empfehlung

- ▶ Qualität ist kontextabhängig
  - Kultur, Prägung, Erfahrung, ...
  - Konkrete Aufgabe
  
- ▶ Grundkonzepte
  - Haben sich in mehr als einem Kontext bewährt
  - Können vor manchen Stolperfallen bewahren
  
- ☞ Von anderen lernen, ohne sich (zu sehr) einzuschränken

### Bewährtes vs. Innovation

- ▶ „Best/good practices“ sind nie „State of the art“.
- ▶ Abwägung zwischen Bewährtem und neuen Lösungen

### Zwei Sichtweisen auf eine GUI

#### 1 Anwendersicht

- Oberfläche ist „Mittel zum Zweck“
- Ergonomie („Benutzerfreundlichkeit“) steht im Mittelpunkt

#### 2 Entwicklersicht

- Robuster, modularer, wiederverwertbarer Code
- Einfach zu pflegen und zu erweitern

☞ Beide Sichtweisen werden nachfolgend vorgestellt.



### Berücksichtigung ergonomischer Grundsätze bei Software

- ▶ Empfehlungen in der Norm DIN EN ISO 9241 geregelt

### Grundsätze der Dialoggestaltung nach DIN EN ISO 9241-110

- 1 Aufgabenangemessenheit
- 2 Selbstbeschreibungsfähigkeit
- 3 Steuerbarkeit
- 4 Erwartungskonformität
- 5 Fehlertoleranz
- 6 Individualisierbarkeit
- 7 Lernförderlichkeit



### Aufgabenangemessenheit

*Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine **Arbeitsaufgabe effektiv und effizient** zu erledigen.*

- ▶ Funktionalität und Dialog basieren auf den charakteristischen Eigenschaften der Arbeitsaufgabe, nicht auf der zugrundeliegenden Technik.
  - ▶ Beispiel: Sinnvolle Standardwerte bei Eingabefeldern
  - ▶ Voraussetzung: Analyse der Arbeitsabläufe
- ☛ Programme für Menschen, nicht für Computer

DIN EN ISO 9241-10



### Selbstbeschreibungsfähigkeit

*Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems **unmittelbar verständlich** ist oder dem Benutzer **auf Anfrage erklärt** wird.*

- ▶ Wo bin ich gerade (im Arbeitsablauf)?
- ▶ Was kann ich (als nächstes) tun?
- ▶ Wie kann ich den nächsten Schritt tun?

DIN EN ISO 9241-10



### Steuerbarkeit

*Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den **Dialogablauf** zu **starten** sowie seine **Richtung und Geschwindigkeit** zu **beeinflussen**, bis das Ziel erreicht ist.*

#### ▶ Beispiele

- Aktueller Schritt kann abgebrochen werden.
- Letzter Schritt kann rückgängig gemacht werden.
- Auswahl mehrerer Optionen für den nächsten Schritt

☞ Nutzer zumindest das Gefühl der Kontrolle vermitteln



### Erwartungskonformität

*Ein Dialog ist erwartungskonform, wenn er **konsistent** ist und den Merkmalen des Benutzers entspricht, z.B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung sowie den **allgemein anerkannten Konventionen**.*

- ▶ Konventionen sind lediglich ein Aspekt.
- ▶ Konsistenz erhöht grundsätzlich die Vorhersehbarkeit.
- ▶ Beispiele
  - Identische Tastenkürzel in allen Fenstern
  - Tabulatorreihenfolge entspricht dem Arbeitsablauf

DIN EN ISO 9241-10



### Fehlertoleranz

*Ein Dialog ist fehlertolerant, wenn das beabsichtigte **Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben** entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.*

- ▶ Beispiel
  - Überprüfung von Nutzereingaben und Rückmeldung bei Falscheingabe
- ▶ Voraussetzungen
  - Robuster Code
  - Verständliche und korrekte Fehler-/Warnmeldungen

DIN EN ISO 9241-10



### Individualisierbarkeit

*Ein Dialog ist individualisierbar, wenn das Dialogsystem **Anpassungen an die Erfordernisse** der Arbeitsaufgabe sowie an die **individuellen Fähigkeiten und Vorlieben des Benutzers** zulässt.*

- ▶ Weitgehende (einfache) Konfigurierbarkeit
- ▶ Beispiele
  - Position und Größe von Fenstern
  - Unterschiedliche Ansichten (normal/Experte)
  - Vorbelegung von Eingabefeldern mit Werten

DIN EN ISO 9241-10



### Lernförderlichkeit

*Ein Dialog ist lernförderlich, wenn er den Benutzer beim **Erlernen des Dialogsystems unterstützt und anleitet.***

#### ▶ Beispiele

- „Guided Tour“ durch die Anwendung
- Eingebaute Hilfefenster
- *Tooltips* für die einzelnen GUI-Elemente
- Intuitive Anordnung der Elemente

#### ▶ Voraussetzung: Gute Kenntnis der Arbeitsabläufe

☞ Aber: Lieber intuitive Anordnung statt Dokumentation



### Einige hilfreiche Programmierkonzepte

- ▶ There is more than one way to do it (TIMTOWTDI)
- ▶ Keep easy things easy and the hard possible
- ▶ Don't repeat yourself (DRY)
- ▶ Keep it simple stupid (KISS)
- ▶ Convention over configuration
- ▶ Aussehen, Programmlogik und Datenverarbeitung trennen
- ☞ Generelle Konzepte für robusten, einfach wartbaren Code
- ☞ Lassen sich gewinnbringend für GUIs einsetzen



### There is more than one way to do it (TIMTOWTDI)

- ▶ Idee
  - Viele Wege führen nach Rom.
  - Keine unnötige Beschränkung der Anwender
  - Was einfach ist, ist oft vom Anwender abhängig.
- ▶ Herkunft
  - Grundprinzip der Programmiersprache Perl
- ▶ Erweiterung
  - There's more than one way to do it,  
but sometimes consistency is not a bad thing either  
(TIMTOWTDIBSCINABTE)
  - Ggf. Konventionen innerhalb eines Projektes festlegen



### Keep easy things easy and the hard possible

- ▶ Einfache Dinge einfach halten
  - „Einfach“ ist ein dehnbarer Begriff.
  - Ziel: Intuitive Verständlichkeit, Offensichtlichkeit
  - Programm schnell in Grundzügen bedienbar
- ▶ Schwierige Dinge ermöglichen
  - Keine Einschränkung der Freiheiten des Nutzers
  - Die Realität ist häufig komplex.
- ▶ Balance zwischen Bequemlichkeit und Freiheit
  - Entscheidung letztlich immer subjektiv
  - Einfache/häufige Abläufe prominent platzieren
  - Modularität hilft bei komplexen oder unvorhergesehenen Abläufen



### Don't repeat yourself (DRY)

- ▶ Grundkonzept der Modularität
  - Eine Aufgabe wird von *genau einer* Routine erledigt.
  - Eine Routine erledigt *genau eine* Aufgabe.
  
- ▶ Vorteile
  - *Genau eine* Stelle im Code muss gepflegt werden.
  - Maximale Flexibilität
  
- ▶ Voraussetzungen
  - Hohes Abstraktionsvermögen
  - Disziplin und kontinuierliches Refactoring



### Keep it simple stupid (KISS)

- ▶ Idee
  - Einfache Systeme sind einfacher zu warten.
  - Oft fehlen die Mittel, komplexe Systeme zu warten.
  - Einfache Systeme sind oft robuster.
- ▶ Herkunft
  - U.S. Navy / Lockheed Skunk Works (u.a. U-2, SR-71)
- ▶ Andere Varianten
  - Weniger ist mehr (Mies van der Rohe)
  - Einfachheit ist die höchste Stufe der Vollendung (Leonardo da Vinci)



### Convention over configuration

- ▶ Idee
  - Vermeidung der Notwendigkeit komplexer Konfiguration
  - Sinnvolle Voreinstellungen, die sich ggf. ändern lassen
- ▶ Voraussetzungen
  - Gute Kenntnis der Anforderungen und Gegebenheiten (Was ist im gegebenen Kontext „sinnvoll“?)
  - Dokumentation der Konventionen
  - Konsequente Einhaltung und Umsetzung
- ▶ Abgrenzung
  - Schließt Konfigurationsmöglichkeiten keinesfalls aus
  - Konventionen nicht notwendigerweise statisch



### Aussehen, Programmlogik und Datenverarbeitung trennen

- ▶ Aussehen
  - Elemente der GUI/CLI und ihre Anordnung
- ▶ Programmlogik
  - Zusammenspiel der einzelnen Komponenten der GUI/CLI
- ▶ Datenverarbeitung
  - Jegliche Operation auf den eigentlichen Daten
- ☞ Ähnlich dem MVC-Konzept (*Model View Controller*)
- ☞ Konkrete Ausgestaltung der Trennung vom konkreten Einzelfall abhängig



### Gründe für die Trennung

- ▶ **Modularität**
  - Eine Aufgabe, eine Routine (Übersichtlichkeit)
  - Wiederverwertbarkeit, Wartbarkeit
- ▶ **Nachvollziehbarkeit**
  - Historie, Versionsnummer der Toolbox, VCS
  - Datenverarbeitung ohne GUI nachvollziehbar
- ▶ **Automatisierbarkeit**
  - Abläufe skriptbar
- ▶ **Unabhängigkeit**
  - Datenverarbeitung ohne GUI (vgl. Nachvollziehbarkeit)
  - Inkompatibilitäten bei GUIs wesentlich wahrscheinlicher



Lounge Chair & Ottoman, Charles & Ray Eames, 1956



### Designentscheidungen – mehrere Ebenen

#### ▶ Anwendersicht

- Sprache der Elemente
- Anordnung von Elementen
- GUI in der Größe veränderbar?
- Ein oder mehrere Fenster
- Verwendung von Menüs
- Tastenkürzel

#### ▶ Entwicklersicht

- Entwicklung: per Hand oder mit Hilfsmitteln
- Programmiersprache und Grafik-Bibliothek
- Plattformunabhängige Entwicklung?

☞ Keinerlei Anspruch auf Vollständigkeit



### Sprache der Elemente

- ▶ **Wissenschaft ist international**
  - Englisch ist in vielen Fällen die Sprache der Wahl.
  - Alternative: Internationalisierung (komplexe Umsetzung)
  
- ▶ **Prägnanz und Eindeutigkeit**
  - Wichtig für schnelles Erfassen der jeweiligen Situation
  - Kurze Begriffe und erklärende Texte
  - Prägnanz durch im Kontext gängige Begriffe
  - Eindeutigkeit immer im Kontext der Aufgabe
  - Erfordert hinreichende Sprachbeherrschung
  
- ▶ **Korrektheit**
  - GUI-Elemente lassen sich vom Anwender selten ändern.
  - Mancher stört sich an (Rechtschreib-)Fehlern.



### Anordnung von Elementen

- ▶ Übersichtlich
  - Wir können nur wenige Elemente auf einen Blick erfassen.
  - Zusammengehöriges zusammen anordnen
  - Zusätzliche (selten genutzte) Funktionen „verstecken“
- ▶ Den Arbeitsabläufen entsprechend
  - Setzt tiefe Kenntnis gängiger Arbeitsabläufe voraus
  - Ergonomie: Optimierung der Schnittstelle auf die Abläufe
- ▶ Erwartungskonform
  - Anordnung der Elemente wie vom Nutzer erwartet
  - Praktikabilität geht vor „Offensichtlichkeit“



### GUI in der Größe veränderbar?

- ▶ **Nutzersicht**
  - Anpassung der Größe oft bequem
  - Bekannt von vielen Programmen
  - Hardware (insbesondere Bildschirmgröße) unterschiedlich
  
- ▶ **Entwicklersicht**
  - Statische Größe wesentlich einfacher implementierbar
  - Größenänderung erfordert klare Behandlungsroutinen für jedes GUI-Element (Größenänderung ist ein Ereignis)
  - Minimale sinnvolle Größe festlegen
  
- ☞ **Lässt sich nur mit erheblichem Zeitaufwand im Nachhinein implementieren**

### Ein oder mehrere Fenster

- ▶ Ein Fenster
  - Alle Elemente in einem Fenster
  - Gliederung durch Registerkarten (Tabs) o.ä.
  - Übersichtlichkeit durch unterschiedliche „Ansichten“
  
- ▶ Mehrere Fenster
  - Meist ein Hauptfenster
  - Ein Fenster für jede (zusätzliche) Aufgabe
  - Kommunikation zwischen Fenstern notwendig
  - Kann unübersichtlich werden (Fenster verdeckt, aber offen)
  - Hauptfenster schließt andere Fenster beim Beenden mit

☞ In der Praxis oft eine Mischung beider Konzepte



### Verwendung von Menüs

- ▶ **Bewährtes Konzept**
  - Seit den Anfängen grafischer Oberflächen bekannt
  - Kann als beim Nutzer bekannt vorausgesetzt werden
  
- ▶ **Übersichtlichkeit**
  - Alle Funktionen auf einen Blick („Inhaltsverzeichnis“)
  - Aber: Oft Zuordnung zu Menüs nicht eindeutig (und damit wenig intuitiv)
  
- ▶ **„Menüfreie“ Systeme**
  - Mitunter intuitivere Nutzerführung
  - Komplexe Aufgaben mit komplexer Nutzerinteraktion schwer in Menüpunkt zusammenzufassen
  - Übersicht ggf. schwerer zu bekommen



### Tastenkürzel

- ▶ GUIs sind für die Bedienung mit der Maus ausgelegt.
  - Zentraler Aspekt der Bedienerfreundlichkeit
  - Gerade für Gelegenheitsnutzer wichtig
  - Schnelle Bedienung durch intuitive Anordnung
  
- ▶ Tastenkürzel
  - Oft schneller als der Griff zur Maus
  - Immer nur Ergänzung zur grafischen Bedienung
  - *Per se* unintuitiv, deshalb: gut dokumentieren
  - Besonders für Vielnutzer geeignet
  - Können Arbeitsprozesse immens beschleunigen

☞ Möglichst intuitive bzw. bekannte Tastenkürzel einsetzen



### Entwicklung: per Hand oder mit Hilfsmitteln

#### ▶ Hilfsmittel

- GUIs zum Design von GUIs
- Code wird automatisch generiert.
- Generierter Code selten so gut wie handgeschriebener
- Generierter Code meist konsistent

#### ▶ Programmierung per Hand

- Deutlich aufwendiger
- Konsistenz Frage der Disziplin des Programmierers
- Vollständige Kontrolle über alle Aspekte

☞ Abhängig von Zielstellung, Komplexität, Ressourcen, ...

☞ Moderne IDEs nehmen Arbeit beim Programmieren ab.



### Programmiersprache und Grafik-Bibliothek

- ▶ **Programmiersprache**
  - Unabhängig von der Grafik-Bibliothek
  - Implementierung der eigentlichen Datenverarbeitung
  - Sollte auf die Problemstellung zugeschnitten sein
  - Möglichst frei verfügbar und plattformunabhängig
  
- ▶ **Grafik-Bibliothek**
  - Unabhängig von der Programmiersprache
  - Viele Bibliotheken haben Bindungen für diverse Sprachen.
  - Möglichst frei verfügbar und plattformunabhängig
  
- ▶ **Spezialfall Matlab**
  - Programmiersprache und Grafik-Bibliothek aus einer Hand
  - Kommerziell

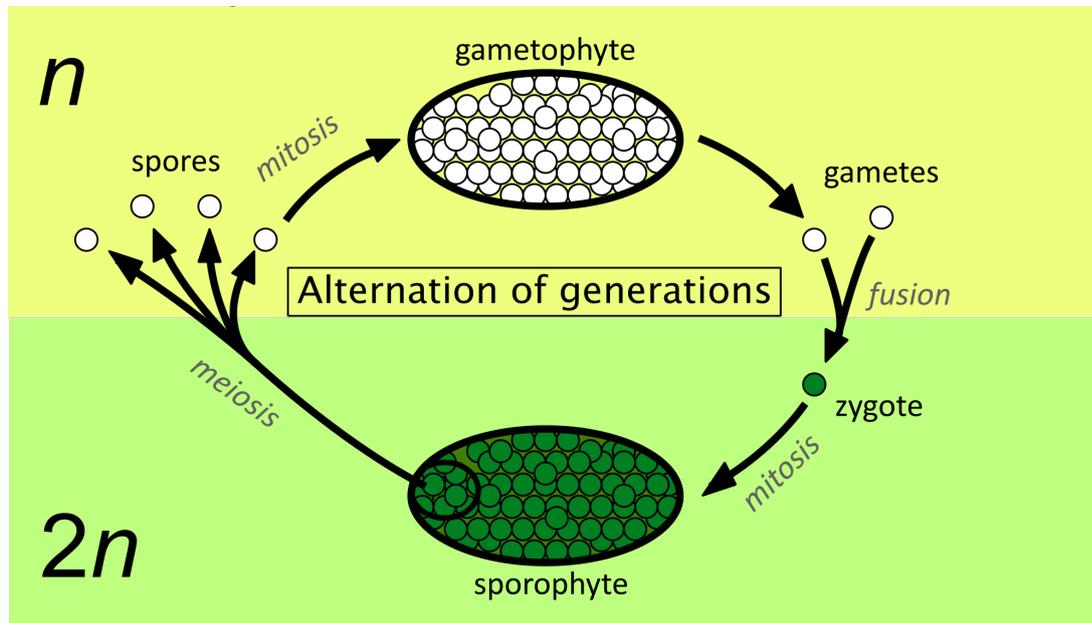


### Plattformunabhängige Entwicklung?

- ▶ Heterogene IT-Landschaft in der Wissenschaft
  - Alle (drei) großen Betriebssysteme vertreten
  - Messsoftware oft auf eine Plattform festgelegt
  - Geräte oft wesentlich längerlebig als Betriebssysteme
  
- ▶ Plattformunabhängigkeit auf Systemebene
  - Programmiersprache und Grafik-Bibliothek
  - Manuelle Tests aufwendig und i.d.R. unrealistisch
  - Abstraktionsschichten aktiv und konsequent nutzen
  
- ▶ Praxis
  - Virtuelle Maschinen
  - Automatisierte Tests
  - Ausreichend großer und heterogener Nutzerkreis

# Allgemeines zur GUI-Entwicklung

## Allgemeiner Entwicklungszyklus einer GUI





### Allgemeiner Entwicklungszyklus einer GUI

- 1 Was soll die GUI können?
  - Anforderungsanalyse (so detailliert wie möglich)
  - Enge Abstimmung mit der (künftigen) Nutzergruppe
- 2 Abläufe überlegen
  - Erfahrungen und Konventionen berücksichtigen
- 3 Prototypen implementieren und optimieren
  - Prototypen verarbeiten keine Daten
  - Programmlogik möglichst vollständig implementieren
  - Abläufe aus der Praxis heraus optimieren
- 4 Funktionen hinterlegen
  - Idealerweise bereits vorhanden
  - Modular und von der GUI unabhängig



### Tipps aus der Praxis

- ▶ **Anforderungsanalyse**
  - Möglichst klar formulierte Zielstellungen
  - Arbeitsabläufe aus der Praxis kommend festhalten
  - Enge Abstimmung mit (künftiger) Nutzergruppe
  
- ▶ **GUI-Design beginnt mit Zettel und Stift.**
  - Wesentlich flexibler als Prototypen
  - Schnellere Entwicklungszyklen möglich
  - Kosten-Nutzen-Rechnung
  - Voraussetzung: Vorstellungskraft, Abstraktionsvermögen
  
- ▶ **Abläufe lassen sich oft erst in der Praxis optimieren.**
  - Setzt (voll) funktionsfähige Prototypen voraus
  - Skaliert mit der eigenen Erfahrung



### Tipps aus der Praxis (Fortsetzung)

- ▶ Auf die Nutzer hören
  - Implementierung macht oft betriebsblind
  - Nutzer steht im Fokus, nicht die dahinterliegende Technik
  
- ▶ Nutzern über die Schulter schauen
  - Wir sind daran gewöhnt, „workarounds“ zu finden.
  - Nutzer verwenden GUIs häufig anders als gedacht.
  - Offenbart häufig Schwächen in der Implementierung
  
- ▶ GUIs als Entwickler an realen Beispielen testen
  - Wenn etwas schon den Entwickler nervt...
  - Ermöglicht deutlich kürzere Entwicklungszyklen
  - Voraussetzung: Satz an (realistischen) Testzenarien



## Worte der Warnung

- ▶ **Matlab ist kommerziell**
  - Matlab-GUIs sind ausschließlich mit Matlab lauffähig.
  - GUI-Programmierung ist aufwendig.
  - Die in Matlab-GUIs investierte Zeit kann nicht direkt auf andere Systeme übertragen werden.
  
- ▶ **Matlab wird ständig weiterentwickelt**
  - Änderungen sind jederzeit ohne Vorwarnung möglich.
  - Keine Kontrolle über die Abwärtskompatibilität
  - Bindung an spezielle Matlab-Versionen (Kostenfaktor!)
  
- ☞ **Es gibt zukunftsfähigere Möglichkeiten als Matlab-GUIs, aber nur mit entsprechenden Programmierfähigkeiten.**



## Alternativen zu Matlab-GUIs

### ▶ Frei verfügbare Programmiersprachen

- Fortran
- C/C++
- Python
- ...

### ▶ Frei verfügbare Grafikbibliothek

- wxWidgets
- GTK+
- Qt
- ...

☞ Setzt Beherrschung sowohl der Programmiersprache als auch der Grafikbibliothek voraus.



### Zwei Herangehensweisen in Matlab

#### 1 GUI zur GUI-Entwicklung (GUIDE)

- (Einfache) Erstellung der GUI mittels GUI
- GUI selbst (Aussehen) in binärer `fig`-Datei
- Generiert Teile des Codes selbstständig
- Wenig Einfluss auf die Details der Programmierung

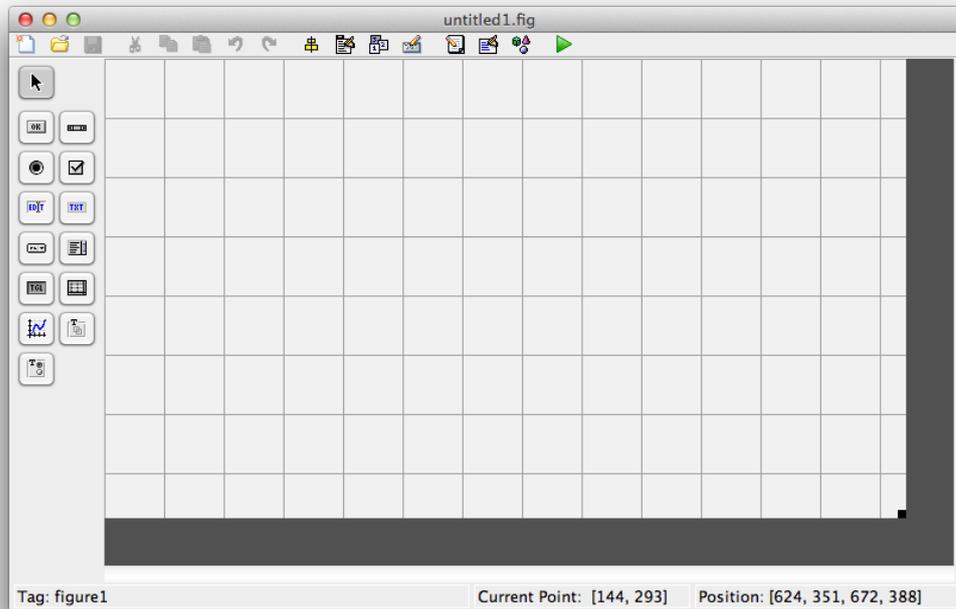
#### 2 Programmatisch (manuell)

- Alle Elemente werden per Hand programmiert.
- Komplettes Aussehen der GUI in einer Funktion
- Voller Zugriff auf alle Details
- Aufwand der manuellen Anordnung der Elemente

👉 Abhängig von persönlichen Vorlieben, Zielstellung, Ressourcen, Komplexität, gegebenem Zeitfenster

# GUI-Entwicklung in Matlab

## Grundsätzliches Herangehen: GUIDE





### Listing 1: Beispiel für die manuelle Programmierung von GUI-Elementen

```
1 hMainFigure = figure('Tag',mfilename,...
2   'Visible','off',...
3   'Name','TA GUI : Main Window',...
4   'Units','Pixels',...
5   'Position',[20,40,950,700],...
6   'Resize','off',...
7   'NumberTitle','off',...
8   'Menu','none','Toolbar','none',...
9   'KeyPressFcn',@keyBindings,...
10  'CloseRequestFcn',@closeGUI);
11
12 % ...
13
14 uicontrol('Tag','export_button',...
15   'Style','pushbutton',...
16   'Parent', hPanelAxis, ...
17   'BackgroundColor',defaultBackground,...
18   'FontUnit','Pixel','FontSize',12,...
19   'String','Detach',...
20   'TooltipString','Export current display to independent Matlab figure',...
21   'pos',[590 0 60 25],...
22   'Enable','off',...
23   'Callback',{@export_pushbutton_Callback}...
24  );
```



### Grundsätzliches Herangehen

- ▶ Man lernt nur an konkreten Beispielen
  - Einfaches Beispiel selbst implementieren
  - Vorher mit Zettel und Stift Konzept entwerfen
- ▶ Matlab-Hilfe
  - Recht ausführliche Einführung in die GUI-Programmierung
  - Referenz für die Eigenschaften von Elementen
- ▶ GUIDE vs. programmatisch
  - Letztlich eine Frage persönlicher Vorliebe
  - Tipp: An konkretem (kleinem) Beispiel ausprobieren
  - Erfahrung: Große Projekte besser programmatisch



### Matlab-GUIs sind Java-basiert

- ▶ GUI beruht (größtenteils) auf Java Swing
    - Mit Matlab programmierbare GUIs und Matlab-GUI selbst
    - Theoretisch alle Elemente der Matlab-GUI verwendbar
    - Matlab-Code zusätzliche Abstraktionsebene
  - ▶ Zugriff auf zugrundeliegende Java-Objekte möglich
    - Setzt u.a. Java-Kenntnisse voraus
    - Ermöglicht Zugriff auf viele zusätzliche Eigenschaften
  - ▶ Kenntnisse von Java zur GUI-Programmierung einsetzbar
    - Zugriff auf zusätzliche GUI-Elemente aus Java-Bibliotheken
    - Änderung des Aussehens und Verhaltens von Elementen
- ☞ Java-Programmierung in Matlab kommt später (kurz)



### Matlab ist ein kommerzielles Produkt

- ▶ Abwärtskompatibilität kein primäres Ziel
  - MathWorks will neue Versionen verkaufen.
  - Mit jeder Matlab-Version neue „Features“
  - Jede neue Version birgt die Gefahr von Inkompatibilitäten.
  
- ▶ Quellcode nicht (bzw. nur in Teilen) offengelegt
  - Jede Menge undokumentierter Funktionen
  - Verwendung undokumentierter Funktionen oft notwendig oder hilfreich, aber risikoreich
  
- ▶ Änderungen jederzeit und ohne Vorwarnung möglich
  - GUIs müssen ständig aktuell gehalten werden.
  - Gilt für GUIs viel mehr als für den sonstigen Code



### Hilfreiche Konzepte

- ▶ Gruppierung von Elementen
  - ▶ Daten in einer GUI ablegen
  - ▶ Status einer GUI speichern
  - ▶ Scripting und Kommandozeile
  - ▶ Hilfefenster in der GUI
  - ▶ Für Fortgeschrittene: Java in Matlab
- 
- ☛ Grundsätzlich (meist) nicht auf Matlab beschränkt
  - ☛ Entstemmen eigener mehrjähriger Erfahrung



### Gruppierung von Elementen

#### ▶ Warum?

- Übersichtlichkeit
- Nutzerführung
- Vereinfachung komplexer GUIs
- Gemeinsames An- und Abschalten (sichtbar/unsichtbar)

#### ▶ Wie?

##### 1 Gruppenfelder (*Panels*)

- Ggf. in Kombination mit Schaltflächen
- Können übereinandergelegt werden
- Schaltflächen zum Wechsel beliebig anzuordnen

##### 2 Registerkarten (Reiter, *Tabs*)

- Bekannt von modernen Browsern
- Offiziell von Matlab nicht unterstützt (undokumentiert)



### Daten in einer GUI ablegen

- ▶ Welche Daten?
  - Datensätze, auf denen gearbeitet wird
  - Informationen über den Zustand der GUI
  - Ggf. Konfigurationseinstellungen für die GUI
  
- ▶ Wie ablegen? – Zwei Möglichkeiten
  - 1 globale Variablen
  - 2 in der GUI selbst – assoziiert mit dem Matlab-Fenster
    - `appdata` beliebige Daten mit Fenster assoziieren
    - `guihandles` Liste der „Handles“ einer GUI
  
- ▶ Hinweise zur Umsetzung
  - Versionsnummer für die `appdata`-Struktur
  - Struktur in eigener Routine zentral definieren



### appdata – Beispiel für die Felder auf oberster Ebene

- ▶ `data` – Datensätze
- ▶ `configuration` – Konfiguration
- ▶ `control` – Zustand der GUI
  - `spectra` – aktiv, inaktiv, modifiziert, ...
  - `measure` – Messung von Punkten, Abständen, ...
  - `axis` – Achseneinstellungen
  - `system` – Informationen zum System
  - `dirs` – Verzeichnisse (Speichern, Laden, ...)
  - `mode` – Modus der GUI (scroll, scale, zoom, CMD, ...)
  - `cmd` – Informationen zur Kommandozeile (CMD)
  - `format` – Format der appdata
  - `status` – Statusmeldungen der GUI



### Zustand einer GUI speichern

- ▶ Idee
  - Aktuellen Zustand inkl. aller Datensätze speichern
  - Später auf genau diesem Zustand weiterarbeiten
- ▶ Schwierigkeiten
  - Zustand kritisch von der Version einer GUI abhängig
  - Inkompatibilitäten können undefinierte Zustände erzeugen.
- ▶ Voraussetzungen
  - Versionierung der appdata-Struktur
  - Routine zur Sicherstellung der Abwärtskompatibilität
- ▶ Format
  - Im einfachsten Fall Matlab-Binärformat
  - Alternative: XML



### Scripting und Kommandozeile

- ▶ Idee
  - Automatisierung von Bearbeitungsschritten
  - Steuerung der GUI (in Teilen) über Skripte
  - Kurzbefehle für komplexere Arbeitsabläufe
  
- ▶ Voraussetzungen
  - Trennung von Programmlogik und Anzeige
  - Konkret: separate Ereignisbehandlungsroutinen
  - Trennung von Programmlogik und Datenverarbeitung
  
- ▶ Hinweise zur Umsetzung
  - Befehle sauber dokumentieren
  - Hilfe zu Befehlen integrieren



### Hilfenfenster in der GUI

- ▶ Idee
  - Hilfe zur GUI in einem eigenen Fenster
  - Einfach zu erreichen (Shortcut, Symbol)
- ▶ Umsetzung
  - Eigenes Fenster
  - Kontextsensitiv (je nach aktiver Ansicht der GUI)
  - Hilfetexte in externen Dateien
- ▶ Erweiterung: Hilfetexte im HTML-Format
  - Voraussetzung: Zugriff auf Java-Elemente
  - Vorteil: Hilfetexte gleichzeitig in Matlab-Hilfe verwendbar

### Für Fortgeschrittene: Java in Matlab

- ▶ Warum? – Beispiele
  - HTML-Hilfefenster
  - Formatierung von Text
  - Cursorposition im Textfeld (Beispiel: Statusfenster)
- ▶ Wie?
  - `findjobj` (File Exchange)
  - Java-Objekt direkt definieren



### Blog von Yair Altman

<http://undocumentedmatlab.com/>

### Textbasierte Nutzerschnittstellen (CLI)

- ▶ Relativ einfach zu implementieren.
- ▶ Lineare (festgelegte) Nutzerführung
- ▶ Notwendigkeit, alles textlich zu beschreiben

### Grafische Nutzerschnittstellen (GUI)

- ▶ Oft schnellerer Zugang für den Gelegenheitsnutzer
  - ▶ Wesentlich aufwendiger in der Programmierung
  - ▶ Matlab: Festlegung auf ein kommerzielles Programm
  - ▶ Plattformunabhängigkeit schwer zu gewährleisten
- ☞ Klare Abwägung: Lohnt sich der Aufwand wirklich?



*So long, and thanks for all the fish.*

## Vorschau: [Toolboxen](#)

- ▶ Struktur von Matlab-Toolboxen
- ▶ Infrastruktur  
(Konfiguration, Installationsroutine, ...)

## Vorschau: [Programmentwicklung](#)

- ▶ „Putting it all together“

Douglas Adams