# Architektur: Kapseln und Auslagerung in Komponenten

| | | | |
|---|---|---|---|
| Fach: | Einführung in die Medizininformatik | Professor: | Erich Schneider |
| Studiengang: | Medizininformatik | Datum: | 4. Juli 2019 |

## Einleitung

Architektur ist die Zerlegung eines Systems in seine Komponenten und eine Beschreibung der Schnittstellen dazwischen. Sie basiert auf bzw. nutzt als Input eine Beschreibung der Anforderungen (an die Software). Der Output ist ein detailliertes (Software-) Design.

## Anforderungen

Ein Python-Programm soll bis zur Prüfung implementiert werden, das folgende Anforderungen erfüllt:

| ID | Beschreibung | Kommentar |
|---|---|---|
| 5 | Nach dem Programmstart werden zwei Registerkarten angezeigt. Die erste Registerkarte (Tab) enthält Funktionen zum Anzeigen und Abspielen von Videos (Videoplayer) und die zweite ein Datendiagramm mit Ergebnissen der Videoanalyse. | |
| 6 | Die Videoanzeige (Videomonitor) zeigt nach dem Start ein verrauschtes Bild mit einer Größe von 10x10 Pixel | |
| 10 | Eine Videodatei im pgm-Format wird interaktiv ausgewählt und geöffnet. Andere Dateiformate sind nicht selektierbar. | |
| 15 | Danach können auch andere pgm-Videodateien beliebig oft ausgewählt und geöffnet werden | |
| 20 | Nach dem Öffnen wird immer die Registerkarte mit dem Video im Vordergrund sowie das erste Bild der Videodatei im Videomonitor dieser Registerkarte angezeigt | |
| 25 | Die Größe des Videomonitors ermöglicht die Betrachtung von Bilddetails aus einer Entfernung von 3 m. | ToDo |
| 30 | Mit einem Schieberegler kann jedes beliebige Bild der Videodatei ausgewählt und im Videomonitor angezeigt werden | |
| 40 | Ein Mausklick auf einen Startknopf spielt die Videodatei ab der aktuellen Position bis zum Erreichen des letzten Bildes ab | |
| 45 | Die Bildauswahl per Schieberegler und das Abspielen der Videodatei kann beliebig oft wiederholt werden | |
| 50 | Am Dateiende stoppt die Verarbeitung | |
| 55 | Beim Abspielen und auch bei der Bildauswahl mittels Schieberegler wird die horizontale und vertikale Pupillenposition jedes angezeigten Bildes berechnet. | |

| ID | Beschreibung | Kommentar |
|---|---|---|
| 56 | Ein weiteres Datendiagramm auf dem Videoplayer-Tab zeigt die aktuell berechnete Pupillenposition in einer EKG-ähnlichen Kurve laufend an. | ToDo: Erfordert die Einbindung von pyqtgraph |
| 60 | Der Schieberegler zeigt die aktuelle Bildposition relativ zum Anfang und Ende der Datei an. | |
| 70 | Die zweite Registerkarte zeigt drei vertikal übereinander ausgerichtete und zunächst leere Datendiagramme an | |
| 80 | Die Datendiagramme stellen nach dem Stop des Abspielevorgangs die Ergebnisse der Video- und Datenanalyse grafisch dar | ToDo |
| 90 | Die Datendiagramme zeigen auf der x-Achse die Zeit und auf den y-Achsen die analysierten Größen. Die Achsenbeschriftungen kennzeichnen die Größen (internationalisiert) sowie deren Einheiten (im SI-System und in eckigen Klammern) an. | Internationalisierung ist umfangreich; erst einmal nicht notwendig. |
| 91 | Das oberste Datendiagramm zeigt die horizontale und vertikale Pupillenposition in Pixel-Einheiten an | |
| 93 | Das mittlere Datendiagramm zeigt die horizontale und vertikale Pupilleneschwindigkeit an. Die „Zacken“ der schnellen Blicksprünge (Sakkaden) werden mit einem geeignet dimensionierten gleitenden (breiten) Medianfilter entfernt. | |
| 94 | Das unterste Datendiagramm zeigt die Absolutgeschindigkeit inkl. Blicksprung-Zacken an | |
| 100 | Die Datendiagramme können als (Vektor-) Bilddateien in den gängigen Formaten (pdf, svg, ...) abgespeichert werden | |
| 110 | Die Datendarstellung kann vergrößert und verschoben werden (zoom, pan) | |
| 120 | Beim Vergrößern und Verschieben der Datendarstellung bleiben die Zeitachsen aller drei Diagramme synchronisiert | |
| | ... | |

## Komponenten

Bisher haben sich zwei wesentliche Funktionalitäten der Software herausgebildet:

1. Abspielfunktion für Videodateien
2. Anzeige der Ergebnisse der Videobearbeitung

die auch in zwei Komponenten bzw. Python-Module ausgelagert werden können.

### Hauptkomponente

Das Hauptfenster bleibt im Hauptmodul und bildet so die Schaltstelle zwischen den anderen Modulen:

```python
from PyQt5 import QtCore, QtGui, QtWidgets
import sys
import traceback

# Nicht benötigte imports entfernen

# Ausgelagerte Module importieren
from gui6_arch_video import VideoTab
from gui6_arch_analysis import AnalysisTab

class MainWindow(QtWidgets.QMainWindow):

    def __init__(self, **kwargs):
        super(MainWindow, self).__init__(**kwargs)
        self.init_ui()

    def init_ui(self):
        self.tabs = QtWidgets.QTabWidget()
        self.setCentralWidget(self.tabs)

        self.videotab = VideoTab(parent=self)
        self.analytab = AnalysisTab(parent=self)

        self.tabs.addTab(self.videotab, self.tr('Video'))
        self.tabs.addTab(self.analytab, self.tr('Analysis'))
        self.tabs.setTabEnabled(self.tabs.indexOf(self.videotab), True)
        self.tabs.setTabEnabled(self.tabs.indexOf(self.analytab), True)
        self.tabs.currentChanged.connect(self.tab_changed)

        menu_file = self.menuBar().addMenu(self.tr('File'))
        menu_file.addAction(self.tr('Open') + ' ...', self.open_file)
        menu_file.addSeparator()
        menu_file.addAction(self.tr('Exit'), self.close)

    def open_file(self):
        # print("open")
        # Inhalte der Methode open_file wurden ausgelagert
        file = self.videotab.open_file()
        # Video-Tab in den Vordergrund
        if file:
            self.tabs.setTabEnabled(self.tabs.indexOf(self.videotab), True)
            self.tabs.setCurrentWidget(self.videotab)

    def tab_changed(self, index):
        try:
            tab = self.tabs.widget(index)
        except:
            traceback.print_exc()


if __name__ == '__main__':

    app = QtCore.QCoreApplication.instance()
    if app is None:
        app = QtWidgets.QApplication(sys.argv)
    app.references = set()
    app.setStyle('Fusion')

    win = MainWindow()
    app.references.add(win)
    win.show()
    win.raise_()
    app.exec_()
```

Dieses Modul importiert am Anfang zwei Klassen aus jeweils einer anderen Python-Datei.

### Komponente Video

Die Klassen zur Videoanzeige werden in eine eigene Python-Datei gui6_arch_video.py ausgelagert und in der Hauptkomponente als Python-Module importiert:

```
from PyQt5 import QtCore, QtGui, QtWidgets
import pgm
import sys
import traceback
import numpy as np

# Nicht benötigte imports entfernen

class VideoFile():

    def __init__(self, pgmfile=None, **kwargs):
        super(VideoFile, self).__init__(**kwargs)
        self.video = None
        self.width = 50
        self.height = 50
        self.length = 10
        self.img_buffer = None
        self.frame_no = 0
        try:
            if pgmfile:
                self.video = pgm.PGMReader(pgmfile)
                self.width = self.video.width
                self.height = self.video.height
                self.length = self.video.length
                self.img_buffer = self.video.img_buffer
            else:
                self.img_buffer = \
                np.uint8(np.random.rand(self.height,self.width)*255)
        except:
            traceback.print_exc()

    def isend(self):
        return self.frame_no>=self.length

    def nextframe(self):
        self.frame_no += 1
        return self.frame_no

    def seek_frame(self, n):
        if self.video:
            self.video.seek_frame(n)
            self.frame_no = n
            # print("Bild %d" % (n))
        else:
            # print("no file %d" %(n))
            return

class VideoWidget(QtWidgets.QWidget):

    def __init__(self, video, **kwargs):
        super(VideoWidget, self).__init__(**kwargs)
        self.painter = QtGui.QPainter()
        self.init_ui(video)

    def init_ui(self, video):
        self.video = video
        self.setFixedSize(video.width, video.height)
        self.image = QtGui.QImage(video.img_buffer,
                                  video.width,
                                  video.height,
                                  QtGui.QImage.Format_Grayscale8)


    def paintEvent(self, event):
        self.painter.begin(self)
```

```
66          self.painter.drawImage(0, 0, self.image)
67          self.painter.end()
68
69  class VideoTab(QtWidgets.QWidget):
70
71      def __init__(self, **kwargs):
72          super(VideoTab, self).__init__(**kwargs)
73          self.video = VideoFile()
74          self.init_ui()
75
76      def init_ui(self):
77          self.display = VideoWidget(self.video, parent=self)
78
79          self.slider = QtWidgets.QSlider(QtCore.Qt.Horizontal)
80          self.slider.setTickInterval(1)
81          self.slider.setValue(0)
82          self.slider.setRange(0, self.video.length - 1)
83          self.slider.valueChanged.connect(self.display_frame)
84
85          self.btn_play = QtWidgets.QPushButton(self.tr('Play'))
86          self.btn_play.setCheckable(True)
87          self.btn_play.clicked.connect(self.play_clicked)
88
89          self.playback_timer = QtCore.QTimer(self)
90          self.playback_timer.setTimerType(QtCore.Qt.PreciseTimer)
91          self.playback_timer.timeout.connect(self.timerfun)
92          self.play_stop()
93
94          layout = QtWidgets.QVBoxLayout(self)
95          layout.addWidget(self.display)
96          layout.addWidget(self.slider)
97          layout.addWidget(self.btn_play)
98
99      def display_frame(self, n):
100         try:
101             self.video.seek_frame(n)
102             self.display.update()
103         except:
104             traceback.print_exc()
105
106     # Funktion open_file zum Öffnen des Dateidialogs hierher verschieben
107     def open_file(self):
108         file, _ = QtWidgets.QFileDialog\
109         .getOpenFileName(caption=self.tr('Choose␣file'),
110                          filter=self.tr('pgm␣Video␣File') + '␣*.pgm')
111         if file:
112             try:
113                 self.open_video(file)
114             except:
115                 traceback.print_exc()
116         # None oder Dateinamen zurückgeben
117         return file
118
119     def open_video(self, pgmfile):
120         self.video = VideoFile(pgmfile)
121         self.slider.setValue(0)
122         self.slider.setRange(0, self.video.length - 1)
123         self.slider.valueChanged.disconnect()
124         self.slider.valueChanged.connect(self.display_frame)
125         self.play_stop()
126         self.display.init_ui(self.video)
127         self.display.update()
128
129     def play_clicked(self):
130         try:
131             # print('click')
132             if self.playback:
133                 self.play_stop()
134             else:
135                 self.play_start()
136         except:
```

```
137             traceback.print_exc()
138
139     def play_start(self):
140         self.playback = True
141         self.playback_timer.start(0)
142         self.slider.valueChanged.disconnect()
143     def play_stop(self):
144         self.playback = False
145         self.playback_timer.stop()
146         self.slider.valueChanged.connect(self.display_frame)
147
148     def timerfun(self):
149         try:
150             if self.video.isend():
151                 self.play_stop()
152             else:
153                 self.video.seek_frame(self.video.frame_no)
154                 self.slider.setValue(self.video.frame_no)
155                 self.display.update()
156                 self.video.nextframe()
157         except EOFError:
158             self.play_stop()
159         except:
160             traceback.print_exc()
161             self.play_stop()
162
163
164 if __name__ == '__main__':
165
166     class MainWindow(QtWidgets.QMainWindow):
167         """
168         Klasse für Hauptfenster verschlanken und in main-Bereich verschieben
169         Wird nur dann aufgerufen, wenn dieses Skript direkt aufgerufen wird,
170         aber nicht wenn Skript als Modul importiert wird.
171         Dieses Verfahren eignet sich gut zum isolierten Testen von Modulen.
172         Dieses Modul enthält nur die Klassen für die Videodarstellung.
173         """
174         def __init__(self, **kwargs):
175             super(MainWindow, self).__init__(**kwargs)
176             self.init_ui()
177
178         def init_ui(self):
179             # Keine Tabs; VideoTab-Klasse wird direkt eingebunden
180             self.videotab = VideoTab(parent=self)
181             self.setCentralWidget(self.videotab)
182
183             menu_file = self.menuBar().addMenu(self.tr('File'))
184             menu_file.addAction(self.tr('Open') + '␣...', self.open_file)
185             menu_file.addSeparator()
186             menu_file.addAction(self.tr('Exit'), self.close)
187             # Beim Testen immer gleich eine Test-Videodatei öffnen
188             # Das erspart Mausklicks
189             self.videotab.open_video('sakkade.pgm')
190
191         def open_file(self):
192             self.videotab.open_file()
193
194
195     app = QtCore.QCoreApplication.instance()
196     if app is None:
197         app = QtWidgets.QApplication(sys.argv)
198     app.references = set()
199     app.setStyle('Fusion')
200
201     win = MainWindow()
202     app.references.add(win)
203     win.show()
204     win.raise_()
205     app.exec_()
```

### Komponente Analysis

Diese Komponente zeigt zunächst leere Datendiagramme an. Später soll sie die Ergebnisse anzeigen.

```
from PyQt5 import QtCore, QtGui, QtWidgets
import pgm
import sys
import traceback
import numpy as np

from matplotlib.figure import Figure
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as QtCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar
import pandas as pd
import abc

class MPLFigure(QtCore.QObject, Figure):

    def __init__(self):
        super(MPLFigure, self).__init__()

    @abc.abstractmethod
    def init_plots(self):
        pass

    @abc.abstractmethod
    def plot_data(self, data):
        pass


class Figure1(MPLFigure):

    def __init__(self):
        super(Figure1, self).__init__()

        self.canvas = QtCanvas(self)
        self.ax1 = self.add_subplot(311)
        self.ax2 = self.add_subplot(312, sharex=self.ax1)
        self.ax3 = self.add_subplot(313, sharex=self.ax1)
        self.init_plots()

    def init_plots(self):
        self.ax1.cla()
        self.ax1.grid()
        self.ax1.set_title(self.tr('Analysis Report'))
        self.ax1.set_ylabel(self.tr('Pupil Position') + ' [px]')
        self.ax1.set_xticklabels([''])

        self.ax2.cla()
        self.ax2.grid()
        self.ax2.set_ylabel(self.tr('Pupil Velocity') + ' [px/s]')
        #self.ax2.set_xticklabels([''])

        self.ax3.cla()
        self.ax3.grid()
        self.ax3.set_ylabel(self.tr('Absolute Velocity') + ' [px/s]')
        self.ax3.set_xlabel(self.tr('Time') + ' [s]')

    def plot_data(self, data):
        self.init_plots()
        # self.ax1.plot(data.loc[:, 'Time'], data.loc[:, ('Hor','Ver')])
        self.ax1.legend(['Hor', 'Ver'], loc='upper right')
        self.ax1.autoscale(enable=True, axis='x', tight=True)

        self.ax2.legend(['Hor', 'Ver'], loc='upper right')
        self.ax2.autoscale(enable=True, axis='both', tight=True)

#         self.ax3.plot(data['Time'], data['vAbs']
#         , data['Time'], data['vConv']
```

```
#         , data['Time'], data['vFilt']
#         , data['Time'], data['vFFT']
#         )
        self.ax3.legend(['vAbs', 'vConv', 'vFilt', 'vFFT'], loc='upper right')
        self.ax3.autoscale(enable=True, axis='both', tight=True)
        np.interp

        self.canvas.draw()


class AnalysisTab(QtWidgets.QWidget):
    output_ready = QtCore.pyqtSignal(str)

    def __init__(self, *args, **kwargs):
        super(AnalysisTab, self).__init__(*args, **kwargs)
        self.init_ui()

    def init_ui(self):
        self.fig1 = Figure1()

        self.toolbar = NavigationToolbar(self.fig1.canvas, self)

        layout = QtWidgets.QVBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        layout.addWidget(self.toolbar)
        layout.addWidget(self.fig1.canvas)

    def plot_data(self, data):
        self.fig1.plot_data(data)


if __name__ == '__main__':

    class MainWindow(QtWidgets.QMainWindow):
        """
        Klasse für Hauptfenster verschlanken und in main-Bereich verschieben
        Wird nur dann aufgerufen, wenn dieses Skript direkt aufgerufen wird,
        aber nicht wenn Skript als Modul importiert wird.
        Dieses Verfahren eignet sich gut zum isolierten Testen von Modulen.
        Dieses Modul enthält nur die Klassen für die Datendiagramme.
        """
        def __init__(self, **kwargs):
            super(MainWindow, self).__init__(**kwargs)
            self.init_ui()

        def init_ui(self):
            # Keine Tabs; AnalysisTab-Klasse wird direkt eingebunden
            self.analytab = AnalysisTab(parent=self)
            self.setCentralWidget(self.analytab)


    app = QtCore.QCoreApplication.instance()
    if app is None:
        app = QtWidgets.QApplication(sys.argv)
    app.references = set()
    app.setStyle('Fusion')

    win = MainWindow()
    app.references.add(win)
    win.show()
    win.raise_()
    app.exec_()
```

## Aufgaben

Erweitern Sie das letzte Python-Programm durch folgende Funktionalität:

1. Fügen Sie eine weitere Komponente mit einer eigenen Klasse hinzu, in der Sie zunächst Dummy-Ergebnisse (z.B. Rauschen) speichern und verwalten.

2. Implementieren Sie die Anforderung 80, zunächst allerdings nur mit Dummy-Ergebnissen. Halten Sie sich dabei an die Vorgaben für eine gute Architektur.

   (a) Tipp: benutzen Sie ein Pandas-Dataframe als Datenspeicher für die Positions- und Geschwindigkeitsdaten.

3. Implementieren Sie auch alle anderen Anforderungen

## Bild- und Datenanalyse

Bauen Sie die Pupillenanalyse, die wir im Mai durchgeführt haben, in das interaktive GUI-Programm ein. Verwenden Sie dazu keine Schleife, sondern bearbeiten Sie jeweils ein Bild in der Timer-Funktion, speichern Sie die Ergebnisse in der zuvor erstellten Komponente und Erfüllen Sie damit die Anforderung 80 komplett. Zur Erinnerung sei hier das Analyse-Skript noch einmal gezeigt.

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 16 12:20:03 2019

@author: eschneid
"""

import pgm
import matplotlib.pyplot as plt
from scipy import ndimage
import numpy as np

pgmfile = "p00001s001e016c001t001.pgm"

video = pgm.PGMReader(pgmfile)
frames = video[0::1]

co_y = np.arange(0, video.height, 1)
co_x = np.arange(0, video.width, 1)
com = np.full((2,video.length), np.nan, dtype=float)

doplot = False
if doplot:
    fig = plt.figure(figsize=(10,5))

for i, frame in enumerate(frames):
    img = frame['data']
    img[100:-1,0:25] = 100
    #img1 = ndimage.median_filter(img, 1)
    #img1 = ndimage.uniform_filter(img, 3)
    img1 = ndimage.gaussian_filter(img, sigma=1)
    h = ndimage.histogram(img1, 0, 255, 256)
    b = img1 < 80
    b[0,:] = False
    b[-3:,:] = False
    img[b] = 255

    sum_x = np.sum(b, axis=0)
    sum_y = np.sum(b, axis=1)
    sum_n = np.sum(sum_x)
    com[0,i] = np.dot(co_x, sum_x)/sum_n
    com[1,i] = np.dot(co_y, sum_y)/sum_n
    icom = np.round(com[:,i]).astype(int)

    img[:,icom[0]-1:icom[0]+1] = 200
    img[icom[1]-1:icom[1]+1,:] = 200
```

```
47      print(i)
48
49      if doplot:
50  #         ax = fig.add_subplot(len(frames),3, i*3+1)
51  #         ax.imshow(img, cmap='gray')
52  #         ax.set_axis_off()
53  #
54  #         ax = fig.add_subplot(len(frames),3, i*3+2)
55  #         ax.imshow(img1, cmap='gray')
56  #         ax.set_axis_off()
57  #
58  #         ax = fig.add_subplot(len(frames),3, i*3+3)
59  #         ax.plot(h)
60          plt.imshow(img, cmap='gray')
61          plt.title('%d/%d␣x=%1.2f␣y=%1.2f' % (i,video.length,com[0,i],com[1,i]))
62          plt.show()
63  plt.show()
64  plt.plot(com.T)
65  plt.show()
```