

Musterlösungen zum 3. Sem. Informatik

Studiengang WIW/Informatik (Prof. Dr. Udo Heuser)

Letzte Aktualisierung: 17.10.2022

Aufgabe:

Entwerfen Sie für uns eine Windows- bzw. Client-Applikation oder eine Smartphone-App, die nach Eingabe zweier natürlicher Zahlen, den größten gemeinsamen Teiler (GGT) ausgibt (Skript S. 30).

Notwendige Schritte:

1. Erstellung eines Gantt-Diagramms zum Softwareentwicklungsprojekt (Skript S. 33)

Bemerkung: Wir verwenden für das ggT-Softwareprojekt das vorgegebene Gantt-Diagramm des Skripts unverändert

Wichtig: Überlegen Sie sich für weitere Software-Projekte, wie Sie im Team die Arbeitspakete personell und zeitlich aufteilen wollen und welche Meilensteine Sie als Team einhalten müssen oder wollen (Zwischenergebnisse, Projektabnahme, etc.). Dies wird Ihr eigenes Gantt-Diagramm maßgeblich beeinflussen.

2. Erstellung eines Mindestanforderungskatalogs / erweitertes Lastenheft (S. 56)

a. Zielgruppen und daraus abgeleitete Anwendungsfälle

Wir haben für unser ggT-Softwareprodukt folgende Zielgruppen definiert:

Zielgruppen sind einerseits Schüler und Studenten (*Zielgruppe A*), andererseits an der Programmierung von Apps interessierte Personen (*Zielgruppe B*).

Bemerkung: Typischerweise können weitere Zielgruppen zu Software-Produkten ausgemacht werden, wie z.B. „Hotelgast“, „Hoteleigentümer“, evtl. auch „IT-Administratoren“, etc. Insofern wirkt die oben genannte Zielgruppe B leicht konstruiert, soll jedoch im Folgenden immer der Vollständigkeit halber mit betrachtet werden.

Wichtig: Überlegen Sie sich für Ihr eigenes SW-Projekt, welche Zielgruppen tatsächlich Ihr SW-Produkt in welcher unterschiedlichen Art und Weise verwenden werden.

Anwendungsfälle:

- Zielgruppe A: Möchte auf einfache Art und Weise zwei natürliche Zahlen eingeben können, um rasch das Ergebnis hierzu innerhalb der gleichen GUI-Eingabe-/Ausgabemaske zu betrachten.
- Zielgruppe B: Ist darüber hinaus interessiert, unterschiedliche Algorithmen-Implementierungen auszutesten und evtl. daraus abgeleitete unterschiedliche Prozesszeiten und/oder Zusatz-/Debuginformationen zu erhalten.

b. funktionale Anforderungen

- es müssen 2 Eingabetextfelder sowie ein Ausgabefeld bereitgestellt werden
- die Eingabe-Textfelder und das Ausgabefeld müssen so breit sein, dass auch große ganze Zahlen eingegeben bzw. ausgegeben werden können
- Zu allen Ein-/Ausgabefeldern sollen per Bezeichnungsfeld (Label) dem Benutzer angezeigt werden, um welche Ein-/Ausgabewerte es sich handelt
- Die Eingaben werden typischerweise als Zeichenketten (String) aufgenommen und müssen intern – und also transparent für den Benutzer – in ganze Zahlen umgewandelt werden
- Die App soll gegen Falscheingaben robust sein: werden andere Werte statt ganzen Zahlen eingegeben (z.B. Buchstaben-/Zahlenkombinationen), so darf die App nicht abstürzen und soll optional eine Warnung an den Benutzer ausgeben.
- Zur SW-Anwendung sind keine Zugriffsrechte vorgesehen. In nachfolgenden Versionen wäre eine vorgeschaltete Authentifizierungsfunktion vorstellbar, die dazu dienen soll nur die genannten Zielgruppen zuzulassen (optional).
- Beispiel-GUI:

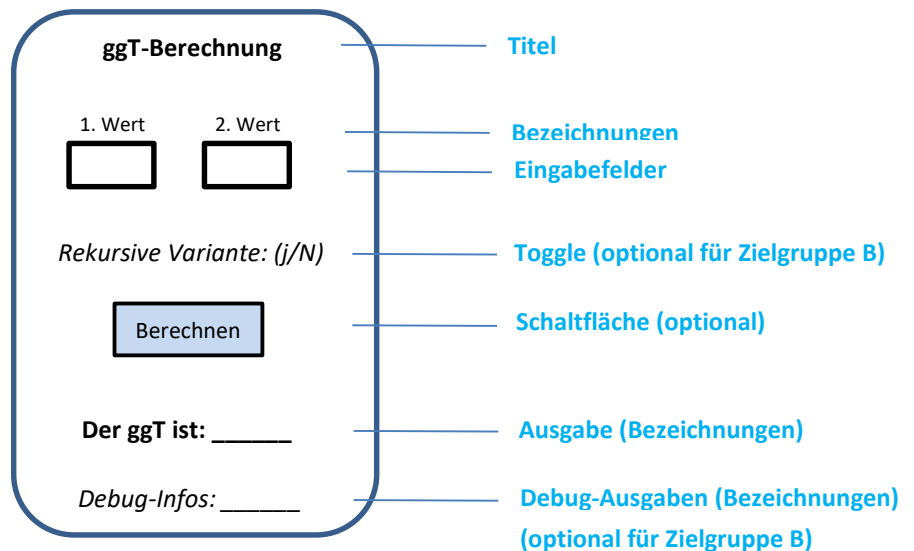


Abb. 1: GUI-Design zum ggT-Softwareprodukt

c. nicht-funktionale Anforderungen

Performance: Die Ergebnisse der Standalone-App müssen dem User in einer Antwortzeit < 0,1 sec angezeigt werden, von Eingabe der Werte bis Ausgabe sollen nicht länger als 5 sec. vergehen.

Anmerkung: Für den Fall weiterer Ausbaustufen dieser App, in der die Business-Logik (und evtl. auch potentielle Datenbank-Ebenen) als Services auf einen separaten Server (Cloud-Dienst) ausgelagert und mit weit höheren Benutzerzahlen gerechnet werden muss, müssen pro User und bei gleichzeitiger Nutzung des Dienstes die Antwortzeiten neu definiert werden. Dies geschieht jedoch typischerweise im Business-Umfeld auf Seiten des Dienstansbieters nach erfolgtem Deployment in Form sogenannter Service-Level-Agreements (SLA).

Wichtig: Diese Performance-Überlegungen, die für professionelle Client-/Server- bzw. Cloud-Umgebungen notwendigerweise angestellt werden müssen, sind für unsere Standalone-App natürlich vollkommen unerheblich. Dennoch ist es immer sinnvoll, sich auch über die Performance (Effizienz oder Antwortzeiten) der eigenen App/Anwendung Gedanken zu machen. Oft stellt sich die Frage der (Leistungs-)Optimierung auch erst, wenn eine erste Version der App/Anwendung fertig entwickelt wurde. Nähere Infos zur Performance-Optimierung finden Expert-Users unter: <https://docs.microsoft.com/de-de/visualstudio/ide/visual-studio-performance-tips-and-tricks?view=vs-2019>

Die Standalone-App ist per se für jeden User 7x24 Std. verfügbar (s. Anmerkungen oben).

Die oben dargestellte GUI garantiert eine optimale Benutzbarkeit sowie durch unsere Open-Source-Strategie und separater Layout-/Markup-Datei eine optimale Anpassbarkeit der App sowie des GUI-Layouts durch externe Software-Experten und Design-Spezialisten.

Die Wartbarkeit ist durch unsere Wahl der Open-Source-Strategie inkl. aussagekräftiger Code-Kommentare sowie einer App-Dokumentation (Doc-Strings) gegeben.

Python-Kodierrichtlinien PEP8 (<https://www.python.org/dev/peps/pep-0008>)

werden weitestgehend beachtet, in Teilen jedoch auf unsere Bedürfnisse angepasst.

d. sonstige Anforderungen (optional)

Die App soll auf einem aktuellen mobilen OS lauffähig sein (z.B. Android 10 und/oder iOS 13.5.x und/oder Windows 10 Mobile; Stand: 6/2020), zumindest jedoch innerhalb des Emulators der eingesetzten IDE). Optional sollen nicht nur die heute aktuellen OS-Versionen, sondern auch mobile OS-Versionen von bis zu vier Jahren vor dem Go-Live des Software-Produkts unterstützt werden (z.B. bis einschließlich *Android 7.0.x/7.1.x Nougat*; Stand: 6/2020). Dies beeinflusst die Hardware-Ausstattung der Smartphones folgendermaßen:

Alle Smartphone-HW von den heute aktuellen HW bis zu vier Jahren ab Go-Live sollen unterstützt werden (optional).

Als Standalone-App ohne notwendige Internet-Konnektivität und kritische Benutzerdaten-Verarbeitung sind die Themen IT-Sicherheit und Datenschutz nicht besonders zu betrachten.

Durch unsere Open-Source-Initiative werden alle ethischen Anforderungen grundsätzlich abgedeckt. Der Code soll auch in Zukunft durch alle Interessierte jederzeit einsehbar sein. Eine Lizenzierung (z.B. GNU Public Licence) ist derzeit nicht vorgesehen.

e. Muss-, Wunsch- und „Out-of-Scope“-Funktionalitäten (Abgrenzungskriterien).

Sämtliche oben bezeichnete und nicht als optional gekennzeichnete funktionale, nicht-funktionale und sonstige Anforderungen sind als MUSS-Anforderungen zu betrachten. Ein Nicht-Erfüllen dieser MUSS-Anforderungen verhindert automatisch die Kundenabnahme, es sei denn, mit dem Kunden werden nach LH-Erstellung Änderungen in diesem vertraglich bzw. schriftlich (*nicht mündlich!*) festgehalten. Sämtliche *optionalen* funktionalen, nicht-funktionalen und sonstigen Anforderungen sind als WUNSCH-Anforderungen zu betrachten. Diese können auch erst in späteren Produkt-Versionen implementiert werden und verhindern *nicht* die Kundenabnahme.

Als Abgrenzungskriterien sind vorstellbar (erweiterbar):

- Zusätzliche Berechnung des kgV
- Vergleich und Ausgabe der Prozesslaufzeiten bei unterschiedlichen Eingabewerten
- Historisierung (Aufzeichnung und Bereitstellung) der Benutzerein- und Ausgaben
- Automatische Verifizierung der Ergebnisse durch Website-Lookup (z.B. über <http://www.openwebschool.de/06/ma/>)
- ...

3. Erstellung Pflichtenheft inkl. Software-Modelle (S. 74)

Bemerkung: Wir haben uns bereits in Abschnitt 2 dazu entschieden, eine Smartphone App-Anwendung zu erstellen. Eine Client-Anwendung wird nicht angestrebt.

Die Smartphone-App-Entwicklung soll auf möglichst vielen mobilen OS lauffähig sein (iOS, Android). Details hierzu im vorherigen Abschnitt.

Als App-Flow-Chart verwenden wir die Graphik auf S. 73 des Skripts.

Zudem ergänzen wir die Epic „GGT“ mit drei User Stories folgendermaßen:

1. User Story: Als ein*e gewöhnliche*r App-Benutzer*in möchte ich auf einfache Art und Weise zwei natürliche Zahlen eingeben können, um nach Betätigung einer Schaltfläche das Ergebnis in Form des GGT dieser Eingabewerte zu sehen.
2. User Story: Als ein*e gewöhnliche*r App-Benutzer*in möchte ich zudem über eine weitere Schaltfläche vorherige Ein- und Ausgaben löschen
3. User Story: Als Programmierer*in möchte ich zusätzlich Debug-Informationen (Zusatzausgaben) sowie Informationen darüber erhalten, welche GGT-Algorithmus-Variante verwendet wurde.

4. Erstellung einer SW-Architektur (S. 83)

- a. Wir nehmen die SW-Architektur auf S. 79 des Skripts als Vorlage und ergänzen bzw. verändern diese entsprechend.
- b. In unserem Beispiel wird auf eine Datenbank-Ebene verzichtet. Die 3-Tier-Architektur verkürzt sich also auf eine 2-Tier-Architektur. Dies geschieht, da zur Berechnung des ggT auf keine größeren Datenbestände zugegriffen werden muss. Alle benötigten Daten stammen vom Benutzer selbst, welche über das (Graphical) User Interface (GUI) als HMI eingegeben werden.

Jedoch wollen wir die GUI selbst aufteilen in die beiden Komponenten

„Eingabemaske (GUI)“ und „Layout“, um beide voneinander getrennt zu halten. So lässt sich das GUI-Layout nach Kundenbedarf auch im späteren Projektverlauf noch ändern (s. Abschnitt 2c: nicht-funktionale Anforderungen).

- c. Anmerkungen: Die beiden Ebenen können in komplexeren Anwendungen auf verschiedene Hosts übertragen werden. So lässt sich die logische Ebene auf einen Applikationsserver, die Datenbankebene auf einen Datenbankserver verteilen. Die GUI verbleibt als einzige Komponente auf den jeweiligen Clients und kommuniziert dabei mit dem darunter liegenden Applikationsserver. Die Unit- und Blackbox-Tests gehören nicht zur eigentlichen SW-Architektur, werden im unteren UML-Diagramm der Vollständigkeit halber dennoch dem Business-Logik-Paket als Teilkomponente hinzugefügt.

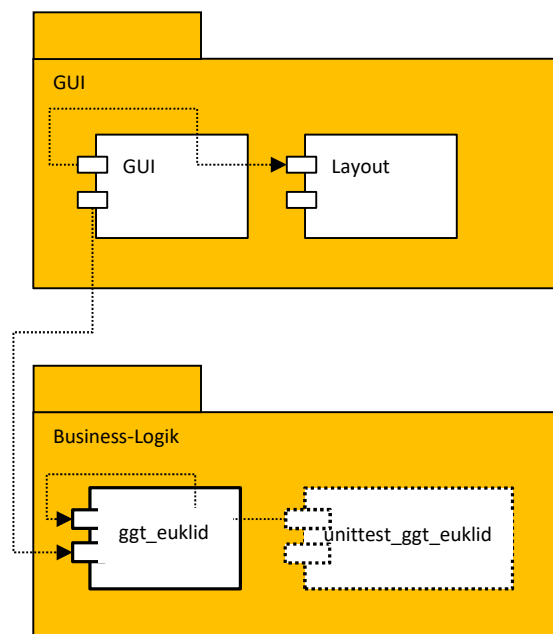


Abb. 2: SW-Architektur für das ggT-Softwareprodukt

5. Erstellung eines Algorithmus' (zur Berechnung des ggT)
 - a. Euklidischer Algorithmus in linearer Variante (S. 89)
 - b. Euklidischer Algorithmus in rekursiver Variante (S. 90)

Bemerkung: Wir verwenden zur Veranschaulichung beide vorgegebene Algorithmen.

Wichtig: Sollten Sie Algorithmen aus externen Quellen verwenden, so geben Sie neben der Darstellung des Algorithmus' die Quelle (URL, Literatur) an.

Wir fügen zu den beiden Algorithmen noch die Vor- und Nachbedingungen hinzu:

- Vorbedingung: die beiden Eingabewerte müssen vom Typ ganze Zahl sein (integer)
- Nachbedingung: Der Ausgabewert muss vom Typ ganze Zahl sein (integer)

6. Implementierung des Software-Modells mit Hilfe von Python und Visual Studio 2019 (S. 123)

a. Euklidischer Algorithmus (linear):

```
def euklid(a, b):
    if a == 0:
        return b
    else:
        while b != 0:
            if a > b:
                a = a - b
            else:
                b = b - a
        return a
```

b. Euklidischer Algorithmus (rekursiv):

```
def euklid_recursive(a, b):
    if b == 0:
        return a
    else:
        return euklid_recursive(b, a % b)
```

c. Modellierung der verwendeten Klassen in UML (optional)

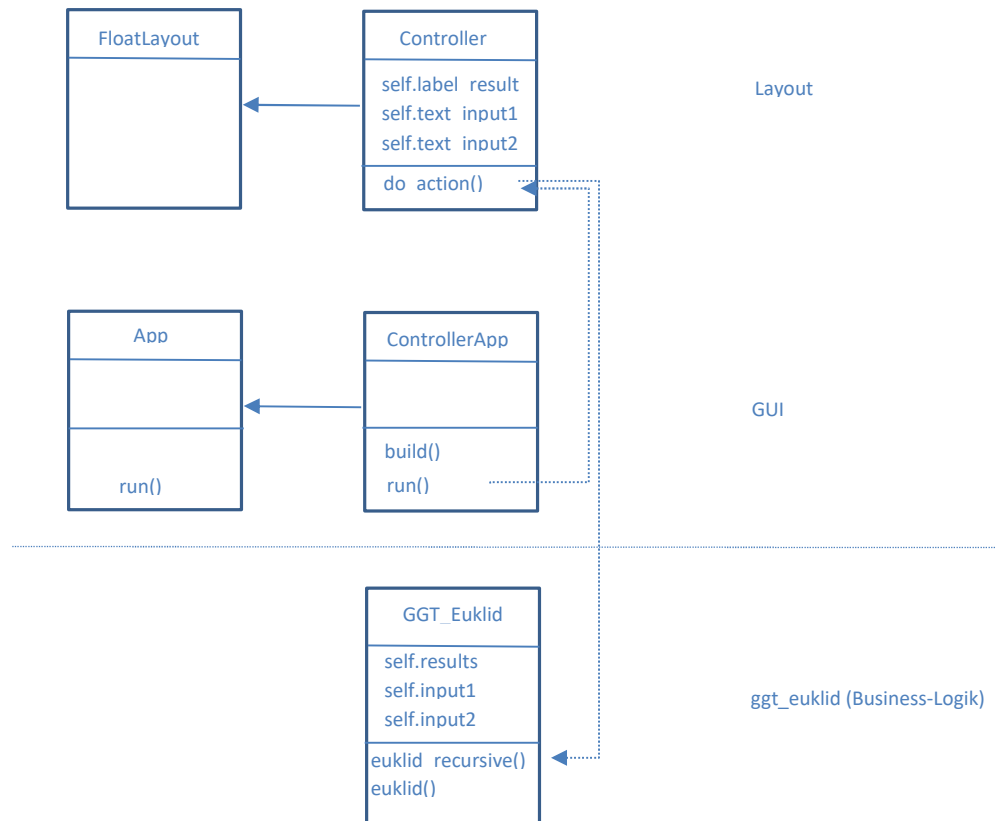


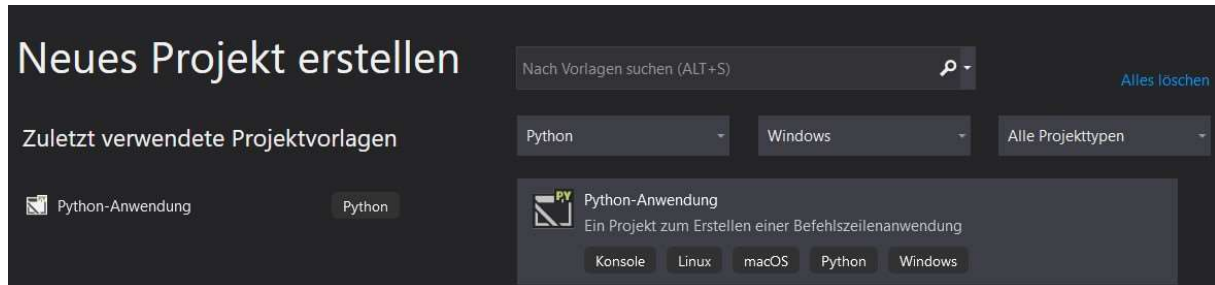
Abb. 3: UML-Klassendiagramme. Hierbei bilden die vier oberen UML-Klassendiagramme die GUI- und Layout-Komponente der Client-Seite, die unter UML-Klasse die Business-Logik des Middle- oder Enterprise-Tier ab.

d. Einrichten der Visual Studio 2019

Bemerkung: Die Einrichtung (Konfiguration) der Visual Studio 2019 (VS) als IDE wird hier der Vollständigkeit halber wiedergegeben. Sie ist nicht Bestandteil eines Pflichtenheft. Ein Auftraggeber erwartet, dass Sie als Software-Dienstleister und Auftragnehmer die gängigsten Software-Tools beherrschen!

Wichtig: In Ihrer Projekt-Dokumentation dürfen Sie gerne auf die Erläuterungen zur Einrichtung Ihrer IDE verzichten, es sei denn, Sie sind zur Verdeutlichung von Aussagen in anderen Abschnitten unbedingt erforderlich!

Anlegen eines Projekts mit Namen „1_ggt_euklid“ über Startbildschirm > Neues Projekt erstellen > Python-Anwendung (Windows, Alle Projekttypen).

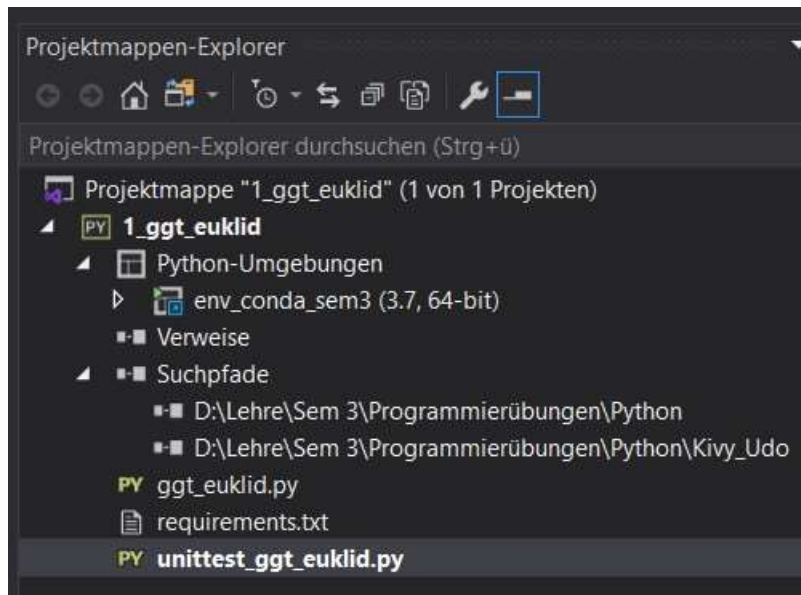


- Projektname: 1_ggt_euklid
- Ort: D:\Lehre\Sem 3\Programmierübungen\Python\Kivy_Udo
- Projektmappe: Neue Projektmappe erstellen
- Projektmappe und Projekt im gleichen Verzeichnis: ja

Dieses Projekt soll später u.a. zwei Python-Module „ggt_euklid.py“ und „unittest_ggt_euklid.py“ beinhalten. „ggt_euklid“ wird die lineare und rekursive Implementierungs-Variante des Algorithmus, „unittest_ggt_euklid.py“ die Debug-Codes und Unittests zum Modul „ggt_euklid“ beinhalten.

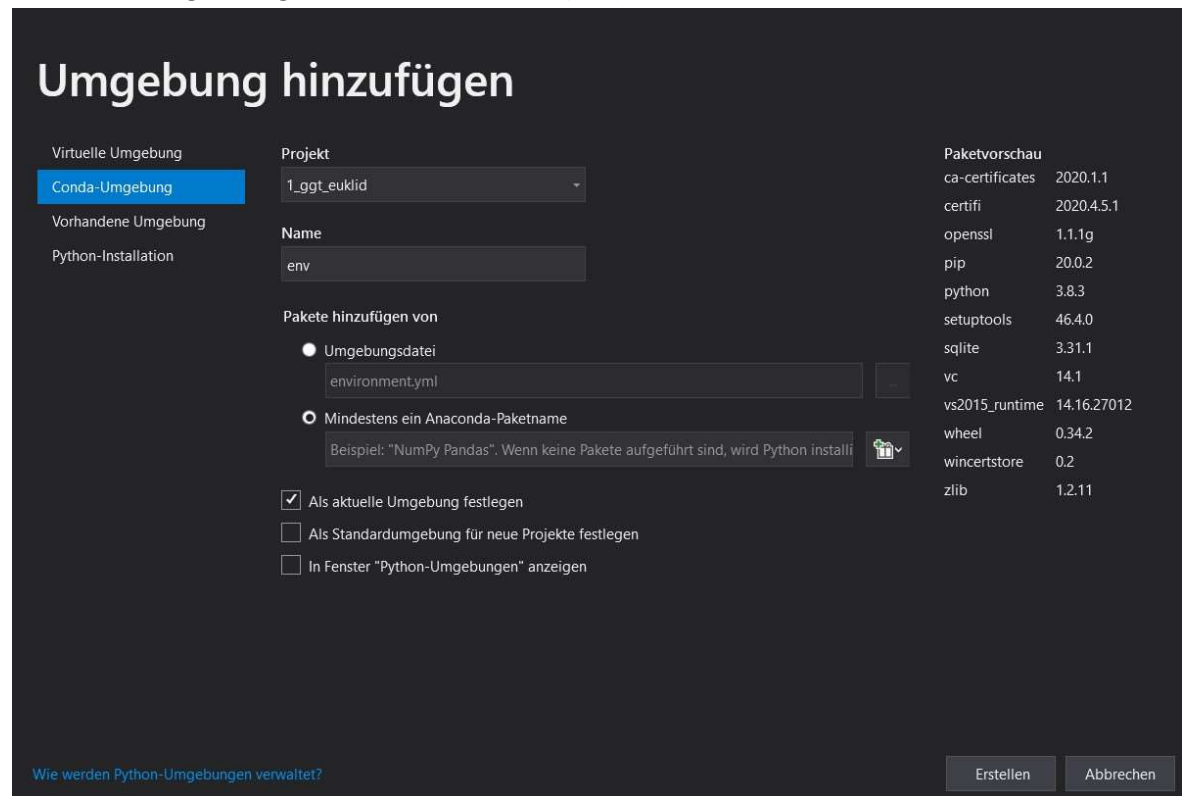
Wichtig: Wir nummerieren die Projektnamen durch, um ein schnelles Auffinden zu ermöglichen. Die hinzuzufügenden Module dürfen dabei nicht durchnummeriert werden, da Module nicht mit einer Zahl beginnen dürfen.

Als Ort wählen wir ein aussagekräftiges Unterverzeichnis in unserem Home-Verzeichnis.

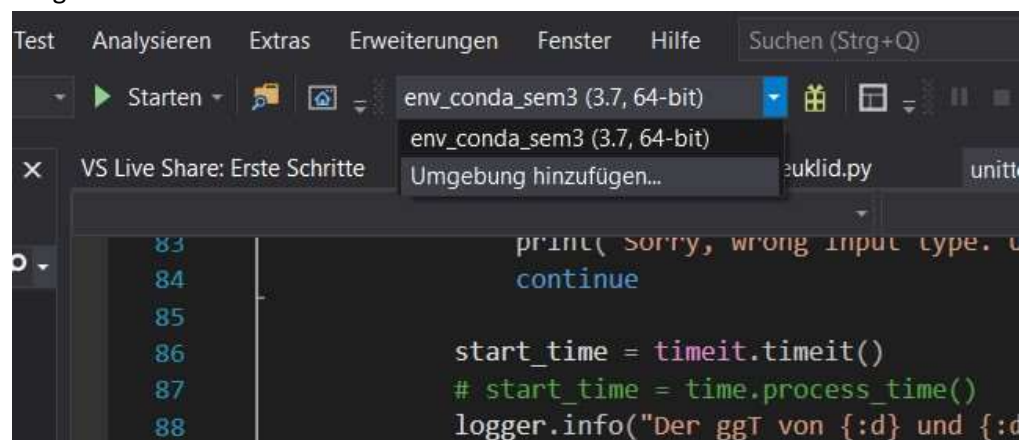


Als Python-Umgebung sollte idealerweise eine Projekt-spezifische Umgebung gewählt werden. Dies geschieht über Rechtsklick auf Projektmappen-Explorer > Python-Umgebungen > „Umgebung hinzufügen > Conda-Umgebung (Anaconda)“.

Dabei sollte zumindest Python in der neusten stabilen Variante (Python 3.7.; Stand: 6/2020) als Paket ausgewählt werden. Weitere Pakete (Import-Module) können später hinzugefügt werden. Zudem sollten in der Definition der neuen Umgebung ausgewählt werden: „Als aktuelle Umgebung festlegen“ und „in Fenster Python-Umgebungen anzeigen“. Als Name ist „env“ durch „env_conda_sem3“ oder einen anderen aussagekräftigen Namen zu ersetzen (s.u.).



Als Default-Umgebung kann auch Anaconda in der neuesten Version (z.B. Anaconda 2019.03 oder Anaconda3 (3.7, 64bit)) ausgewählt werden. Anaconda enthält die relevantesten statistischen und Datenanalyse-Pakete inkl. „kivy“. Im nächsten Schritt kann die so neu generierte Umgebung für alle Python-Projekte ausgewählt werden.



Zu einer einmal erzeugten virtuellen (Conda-)Umgebung lassen sich im Nachhinein weitere Pakete installieren (hier: Paket „pyjnius“). Dies geschieht über Klick auf

gelbes Paket-Symbol rechts neben dem Umgebungsnamen in der oberen Abbildung. Danach unter der aktuellen Umgebung (hier: „env_conda_sem3“) nach dem zu installierenden Paket suchen und durch Klick auf „Befehl ausführen: conda install *pyjnius*“ (Conda-Pakete!) nachinstallieren. Es mag vorkommen, dass Pakete nicht in Conda vorhanden sind. Sie erhalten dann in der Ansicht „Ausgabe“ folgende Fehlermeldung:

```

Ausgabe
Ausgabe anzeigen von: Allgemein

----- "pyjnius" wird installiert -----
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with flexible solve.
PackagesNotFoundError: The following packages are not available from current channels:
- pyjnius
Current channels:
- https://repo.anaconda.com/pkgs/main/win-64
- https://repo.anaconda.com/pkgs/main/noarch
- https://repo.anaconda.com/pkgs/r/win-64
- https://repo.anaconda.com/pkgs/r/noarch
- https://repo.anaconda.com/pkgs/msys2/win-64
- https://repo.anaconda.com/pkgs/msys2/noarch
To search for alternate channels that may provide the conda package you're
looking for, navigate to
https://anaconda.org
and use the search bar at the top of the page.
----- Fehler beim Installieren von "pyjnius" -----

```

In diesem Fall sollten Sie versuchen, das gleichnamige Paket mit Hilfe des PyPI-Repositories zu installieren („Befehl ausführen: pip install *pyjnius*“ (PyPI)).

Im Beispiel des Moduls „unittest_ggt_euklid“ müssen Sie folgende Pakete nachinstallieren:

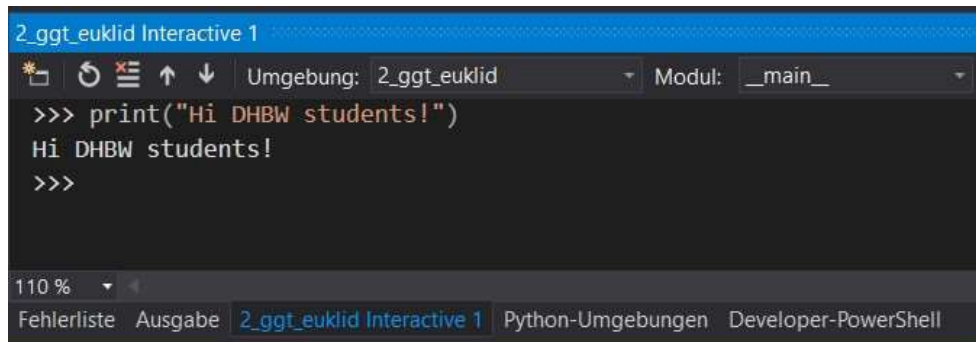
- *pytest* # Alternative zu unittest
- *timeit* # Zeitmessung
- *logging* # Debugausgaben

Anmerkung: Lässt sich weder über Conda noch PyPI ein Paket installieren, so kann dies dennoch auf folgendem Umweg geschehen:

- Rechtsklick auf Projektmappen-Explorer > Python-Umgebungen > env_conda_sem3 > Eingabeaufforderung hier öffnen (bzw. über Ansicht > Python-Umgebungen > Übersicht > env_conda_sem3 > In PowerShell öffnen).
- Innerhalb der Kommandozeile folgenden Befehl eingeben (bei Installation des Moduls kivy): `conda install -c conda-forge kivy`
(Weitere Optionen: <https://anaconda.org/conda-forge/kivy> oder <https://anaconda.org/anaconda/repo> oder Google ;-)

Um den vorliegenden Code zu debuggen sind einige Ausgabefenster besonders nützlich. Diese können über die Ansicht hinzugewählt werden:

- Ausgabe
- Fehlerliste
- Interaktives Fenster (der aktuellen Umgebung) durch Rechtsklick auf Projektmappen-Explorer > aktuelle Umgebung > Interactive Fenster öffnen
- Terminal (Developer-PowerShell)



```
2_ggt_euklid Interactive 1
>>> print("Hi DHBW students!")
Hi DHBW students!
>>>
```

Mit Rechtsklick auf Projektmappen-Explorer > 1_ggt_euklid (aktuelles Projekt) > Hinzufügen > neues Element (bzw. vorhandenes Element) lässt sich eine neue Python-Datei zum aktuellen Projekt hinzufügen.

Mit Rechtsklick auf beliebige Python-Datei innerhalb Projektmappen-Explorer und Auswahl „Als Startdatei festlegen“ wird dieses als Startdatei für das aktuelle Projekt festgelegt. Davor sollte die Datei einfach mit oder ohne Debugging gestartet werden.

Mit Rechtsklick auf Projektmappen-Explorer > Python-Umgebungen > „Anaconda 2019.03“ > Umgebung aktivieren, wird diese statt der „env_conda_sem3“-Umgebung nachträglich aktiviert.

Mit Rechtsklick auf Projektmappen-Explorer > „env_conda_sem3“ (aktuelle Umgebung) > „requirements.txt“ generieren, werden die Requirements (Voraussetzungen bzgl. benötigter Pakete) für das aktuelle Projekt erzeugt. Bemerkung: Dies ist wichtig, um es anderen Entwicklern zu erleichtern, benötigte Zusatzpakete automatisch nachinstallieren zu lassen. Beim Öffnen der Startdatei innerhalb eines Projekts werden so dem Benutzer nachzuinstallierende Pakete automatisch am oberen Bildschirmrand angezeigt, etwa in der Art: „Mindestens

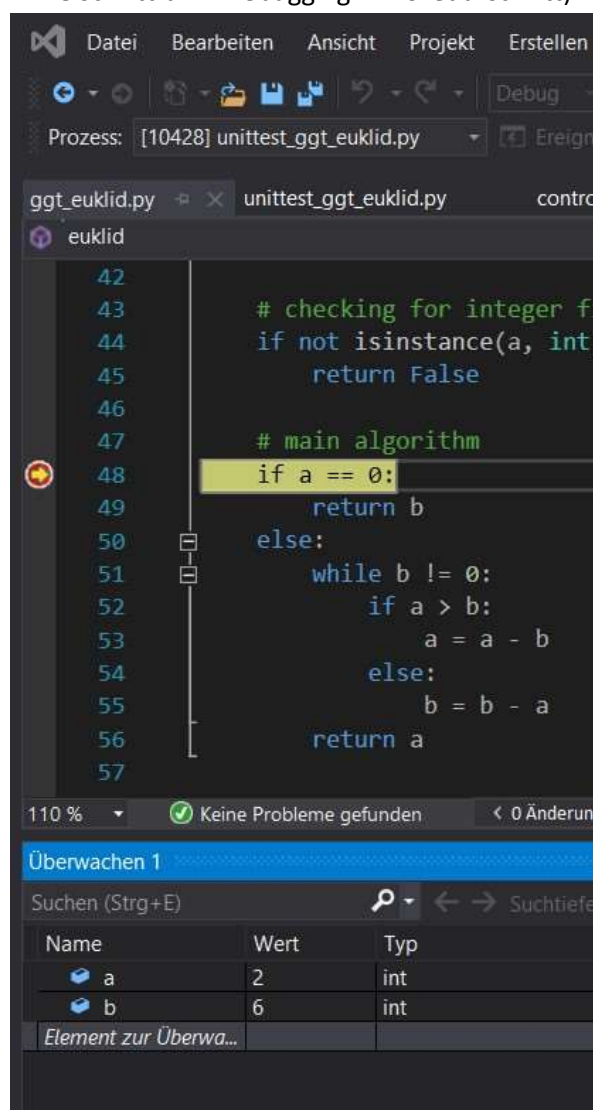
eines der in requirements.txt aufgeführten Pakete des Projekts ExampleProject.md fehlt in der Umgebung ,env_conda_sem3'. **Pakete installieren.**"

Wichtig: Wollen Sie Teile der Projektdateien in einem neuen VS-Projekt wiederverwenden, so erzeugen Sie zunächst über VS ein neues Python-Projekt (vgl. 6c). Anschließend können Sie Projektdateien von dem ursprünglichen Projektordner in den neuen Projektordner *kopieren* und mit „Hinzufügen > vorhandenes Element“ zur VS-Solutions-Datei hinzufügen. Die VS-Solutions-Datei (Endung .sln) und Python-Projektdatei (Endung .pyproj) müssen dabei immer wieder neu erzeugt werden!

7. Erzeugung von Code-Debugging, Blackbox-, Unittests und Vorbedingungen für die Logik-Funktion (S. 156)

a. Code-Debugging

Schon vor Starten eines Moduls kann mit Hilfe von Debuggen > Haltepunkt umschalten ein Haltepunkt (Breakpoint) an beliebiger Stelle im Code gesetzt werden. Der Breakpoint erscheint als roter Punkt links neben dem Code. Anschließend Programm starten. Das Programm läuft bis zum Breakpoint. Ein gelber Pfeil (aktuelle Codezeile der Programmausführung) überlagert den roten Breakpoint. Anschließend über Debuggen > Schnellüberwachung die zu beobachtenden Variablen hinzufügen. Im Beispiel der Funktion `euklid(a, b)` sind vor allem die Werte der beiden Eingabeparameter `a` und `b` interessant. Das Überwachungsfenster kann ermitteln, wie sich die beiden Variablenwerte nach jedem Schritt (Debugging > Einzelschritt bzw. Debugging > Prozedurschritt) verändern.



Anmerkung: Das Code-Debugging dient den Entwicklern dazu, die Code-Ausführung bzw. die Änderung der Variablenwerte zur Laufzeit besser nachvollziehen zu können.

Wichtig: Code-Debugging-Screenshots sind nicht Bestandteil eines professionellen Lasten-/Pflichtenhefts, sollen jedoch an dieser Stelle (und auch in Ihrem Projekt) beispielhaft wiedergegeben werden.

b. Fehlerbehandlung (Beispiele)

Innerhalb der `euklid()` – bzw. `euklid_recursive()` – Methode:

`isinstance()` überprüft, ob eine Variable (als Benutzereingabe) tatsächlich einen Wertebereich (hier: *integer*) entspricht (Vorbedingungen; vgl. Abschnitt 5).

```
# checking for integer first
if not isinstance(a, int) or not isinstance(b, int):
    return False
```

Innerhalb des Moduls `unittest_ggt_euklid`:

`Try .. except` kapselt eine potentiell fehlerträchtige Funktion (hier: `int()`) und gibt eine Warnung an den Benutzer aus anstatt das Programm mit einem Ausnahmefehler zu beenden (Programmabsturz!).

Anmerkung: Die Gefahr liegt darin, dass der Benutzer als Wert keine ganze Zahl eingeben kann. `int()` erwartet jedoch eine Zeichenkette (String), die als ganze Zahl interpretiert und daher umgewandelt werden kann.

```
try:
    a = int(input("Erster Wert: "))
except:
    print("Sorry, wrong input type. Use int instead!")
    continue
```

c. Dokumentation beispielhafter Blackbox- und Unittests (innerhalb des Moduls `unittest_ggt_euklid`)

Unittests:

```
class TestClass:
    # Two test methods, should all be passed ok
    def test_one(self):
        ggt = ggt_euklid.euklid(3, 6)
        assert ggt == 3

    def test_two(self):
        ggt = ggt_euklid.euklid(3, 6)
        assert ggt != 2
```

Blackbox-Tests inkl. Zeitmessung (`import timeit`) und Logging der Ergebnisse

```
(logger.info):
start_time = timeit.timeit()
logger.info("Der ggT von {:d} und {:d} ist (linear):
{:d}".format(a, b, ggt_euklid.euklid(a, b)))
end_time = timeit.timeit()
logger.info("Elapsed time: {:.20.18f} secs".format(end_time - start_time))
```

Mit Hilfe der Blackbox- oder Unittests werden anschließend Tests der Funktionen `euklid(a, b)` sowie `euklid_recursive(a, b)` durchgeführt. Dabei richten sich die Eingabewerte nach den Maßgaben der *Äquivalenzklassenbildung* und *Grenzwertanalyse* von S. 145 des Skripts (hier noch nicht beachtet!). Die Testergebnisse inkl. des erzeugten Teststatus' wird in Tabellenform festgehalten (s.u.). Diese wird typischerweise in einem separaten Testdokument dem Kunden geliefert.

Anmerkung: Der Einfachheit halber veröffentlichen wir die Testergebnisse im gleichen Pflichtenheft.

Test der Funktion <code>ggt_euklid(a, b)</code>					
Testcase	Eingabewert 1	Eingabewert 2	Tatsächliche Ausgabe (IST)	Erwartete Ausgabe (SOLL)	Status
1	3	6	3	3	passed
2	2	4	2	2	passed
3	x	2	Ausgabe auf sys.stdout: "Sorry, wrong input type. Use int instead!"	False (kein Programmabsturz)	(passed)
4	...				

Anmerkung: Im Testcase 3 wird ersichtlich, dass tatsächliche und erwartete Ausgabe nicht unbedingt übereinstimmen müssen, um den Test zu bestehen. In diesem Testfall wird v.a. Wert gelegt, dass das Programm nicht *kommentarlos abstürzt*. Dies ist hier der Fall.

d. Beispielhafte Code-Kommentare (Doc-Strings)

Code-Kommentare und Doc-Strings dienen dazu, den Code auch nach längerer Zeit oder für externe Leser zu verstehen. Mit Hilfe von Doc-Strings lassen sich Code-Manuals (automatisch) erzeugen.

Doc-String zur Klasse TestClass (Modul pytest)

```
"""
Pytest (Einfachere Alternative zu Unittest)
Aufruf aus der Kommandozeile/Shell aus Repository-Verzeichnis des aktuellen VS-
Projekts,
    z.B. D:\Lehre\Sem 3\Programmierübungen\Python\Kivy_Udo\1_ggt_euklid
Parameter -v: verbose
    pytest -v ./unittest_ggt_euklid.py
Contents: https://docs.pytest.org/en/latest/contents.html
Usage: https://docs.pytest.org/en/latest/usage.html
"""
```

Doc-String in Funktion `euklid_recursive`

```
"""Calculate HCF (ggt) using recursive Euklidian algorithm.

Keyword arguments:
a -- first integer number (no default)
b -- second integer number (no default)
Return value:
ggt -- ggt as integer number
Error return value:
False -- if either of the two params are not integer values
"""
```

Code-Kommentar (einzeilig):

```
# error handling wrapping int() function
```


8. Erzeugung der GUI (Smartphone App bzw. Client-UI als Bestandteil der Software-Architektur; S. 78)

Wir entscheiden uns zum einen für *Kivy* als Cross-Plattform-Library zur Entwicklung von touch-fähigen Apps. Und zum anderen für die *KV Language*, um Widgets (Smartphone App-Elemente) in einer zur GUI separaten Datei zu deklarieren (GUI-Layout).

Bemerkung: Professionelle Smartphone-Apps benutzen als GUI-Layout typischerweise Googles *Material Design*. Dies lässt sich durch die Verwendung diverser Libraries in Python realisieren, z.B. über das KivyMD-Layout (<https://github.com/HeaTTheatR/KivyMD>). Daneben bietet auch Anvil an das Material Design angelehnte GUI-Templates an (<https://anvil.works/learn/tutorials/using-material-design>).

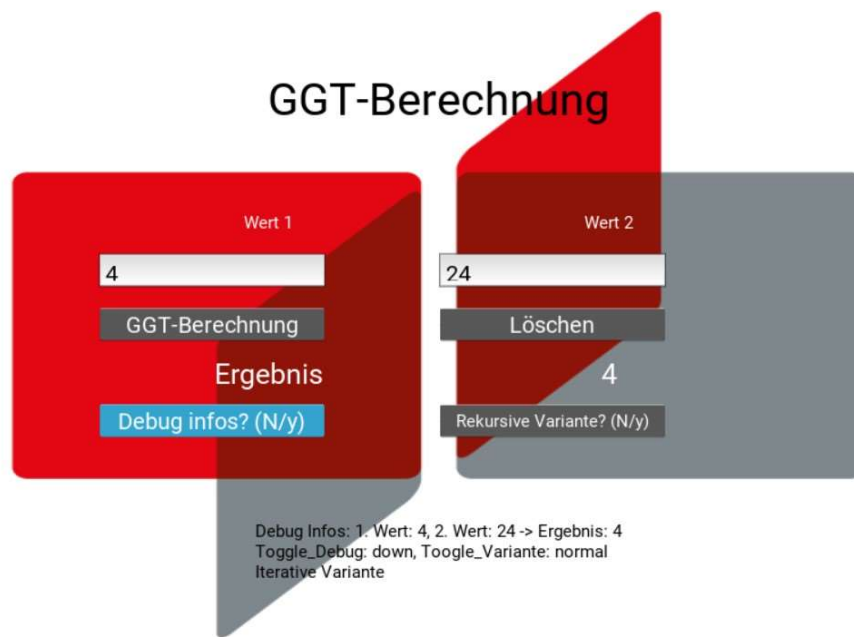
Das KivyMD-Layout muss, im Gegensatz zu den Anvil-Templates, zuvor über das Kommando „conda install kivyMD“ innerhalb der Conda-Umgebung oder „python .\setup.py install“ innerhalb des zuvor entpackten „KivyMD-master“-Verzeichnisses, welches die „setup.py“-Datei beinhaltet, installiert werden.

„KivyMD is a collection of Material Design compliant widgets for use with *Kivy*, a framework for cross-platform, touch-enabled graphical [Python] applications. The project's goal is to approximate Google's *Material Design spec* as close as possible without sacrificing ease of use or application performance. *Kitchen sink* app [contained in the sub-directory of the same name] demonstrates every KivyMD widget. You can see how to use widget in code of app. You can download apk for your smartphone (Android 6.0 and higher): [kitchen_sink-0.102.1-x86.apk](#) or [kitchen_sink-0.103.0-armeabi-v7a.apk](#).“
Weitere Infos: <https://kivymd.readthedocs.io/en/0.104.1/>

Als Alternative und Ergänzung zu Anvil stellen wir im Folgenden die Kivy-Library inkl. der KV Language vor. Sie ist nach einiger Übung für Anfänger ebenfalls leicht zu verwenden. Näheres zu Kivy und KV Language erfahren Sie hier:

- <https://kivy.org/#home>
- <https://kivy.org/doc/stable/guide/lang.html>

In leichter Abwandlung zum GUI-Design der funktionalen Anforderungen aus Abschnitt 2b sieht das Ergebnis folgendermaßen aus:



Anmerkung: Die Schaltfläche „Löschen“ wurde in unserem Beispiel im Laufe der Implementierung der App durch die Programmierer hinzugefügt und ergänzt als „*minor adjustment*“ sinnvoll die bestehenden funktionalen Anforderungen aus Abschnitt 2b.

Wichtig: In realen Projektszenarien wie auch in Ihrem eigenen Projekt sind *größere Anpassungen* jedoch *immer* mit dem Kunden abzusprechen und schriftlich zu fixieren!

Wir definieren hierzu als GUI-Komponente das Modul `main.py` mit folgendem Inhalt (Auszug):

```
class Controller(FloatLayout):

    def do_action(self):

        # Ergebnisberechnung (lineare oder rekursive Variante)

        if EUKLID_RECURSIVE:

            self.label_results.text =
str(ggt_euklid.euklid_recursive(int(self.text_input1.text),
int(self.text_input2.text)))

        else:

            self.label_results.text = str(ggt_euklid.euklid(int(self.text_input1.text),
int(self.text_input2.text)))

        print("Results: ", self.label_results.text)

class ControllerApp(App):

    def build(self):

        return Controller(info='GGT')
```

```

if __name__ == '__main__':
    ControllerApp().run()

```

Erläuterungen: Die Controller-Klasse besitzt im Wesentlichen eine Methode `do_action()`, welche das GGT-Ergebnis auf Basis der Benutzereingaben (Attribute `self.text_input1` und `self.text_input2`) berechnet.

Das Layout wird definiert über die KV-Markup-Definitionen der Datei `controller.kv` (Auszug):

```

#:kivy 1.0

<Controller>:

    text_input1: text_input1

    text_input2: text_input2

    label_results: label_results

    label_debug: label_debug


    canvas:

        Rectangle:

            source: '../DHBW-Raute-Big.png'

            pos: self.pos

            size: self.size

        GridLayout:

            cols: 1

            Label:

                text: root.info

                font_size: '40 sp'

                color: 0,0,0,1 # black


            GridLayout:

                cols: 4


                Label:

                    text: 'Wert 1'

                Label:

                    text: 'Wert 2'

                TextInput:

```

```

        id: text_input1

        size_hint_y: None

        size_hint_x: None

        height: 30

        width: 200

        font_size: self.height - 10
TextInput:

    id: text_input2

    size_hint_y: None

    size_hint_x: None

    height: 30

    width: 200

    font_size: self.height - 10
Button:

    id: btn_result

    text: root.info

    size_hint_y: None

    size_hint_x: None

    height: 30

    width: 200

    font_size: self.height - 10

    on_press: root.do_action()
Button:

    id: btn_clear

    text: 'Clear'

    size_hint_y: None

    size_hint_x: None

    height: 30

    width: 200

    font_size: self.height - 10

    on_press: root.do_clear()

```

. . . .

Die Markup-Datei `controller.kv` bestimmt dabei die Position und das Aussehen aller Widgets (App-Elemente) innerhalb der GUI (Layout) und stößt über `on_press: root.do_action()` die Ergebnisberechnung sowie über `on_press: root.do_clear()` das Löschen bereits existierender Ergebnisse über `main.py` an (vgl. SW-Architektur, Abschnitt 4).

Das Layout selbst kann durch Wahl der Markup-Datei `controller.kv` beliebig geändert werden. Hierzu muss in unserem Beispiel lediglich eine der beiden Markup-Dateien `controller_NestedGrid.kv` oder `controller_PlainGrid.kv`, etc. auf `controller.kv` kopiert werden. Die Layouts lassen sich problemlos um weitere Markup-Dateien ergänzen. Dies entspricht der nicht-funktionale Anforderung aus Abschnitt 2c.

9. Kundenabnahme (S. 160)

Wichtig: In der Kundenabnahme zeigen Sie, dass alle Muss-Kriterien erfüllt und das Software-Produkt lauffähig ist. Dabei zeigen Sie sowohl mit den bereits durchgeführten Unit-/Blackboxtests (s. Abschnitt 7c) bzw. – falls dies nicht möglich ist – GUI-Tests (<https://whatis.techtarget.com/de/definition/GUI-Testing-Test-der-grafischen-Benutzeroberflaeche>) als auch mit live durchgeführten exemplarischen Tests mit beliebigen Eingaben, dass Ihr Produkt tatsächlich das tut, was und wie es vom Kunden gewünscht wurde.

Das Pflichtenheft ist dabei *vor der Präsentation* Ihrer*m Dozenten*in abzugeben. Hierin sollten Sie die *Namen aller Teammitglieder* nennen sowie einen Verweis darauf, wer welche Projektteile verantwortet.

Wichtig: Wenn das Lasten-/Pflichtenheft Ihnen die Wahl lässt, wie Ihre App/Anwendung zu präsentieren ist, so dürfen Sie die App/Anwendung entweder als Emulation über Ihre IDE oder die App auf Ihrem Smartphone zeigen. Im letzten Fall müssen Sie Ihre App zuvor auf dem Smartphone Ihrer Wahl deployen. Zum Deployment einer Kivy-App auf Android-Smartphones gibt es mehrere Möglichkeiten: <https://kivy.org/doc/stable/guide/android.html> bzw. <https://kivy.org/doc/stable/guide/packaging-android.html#packaging-android> oder <https://github.com/damonkohler/sl4a>

10. Aufwandsabschätzung (S. 180)

Wichtig: Die Aufwandsabschätzung wird vor Projektbeginn z.B. mit Hilfe der *Dreipunktschätzung* durchgeführt und an dieser Stelle genannt.

Die Aufwandsabschätzung bestimmt (zusammen mit etwaigen *Lizenzkosten*) die Gesamtkosten Ihres Softwareprodukts. Diese Gesamtkosten können Sie *optional* ebenfalls dem Kunden präsentieren.

Die Aufwandsabschätzung kann *optional* die zusätzlich benötigte Zeit (und evtl. Kosten) für noch offene Anforderungen (*Wunsch-Anforderungen!*) beinhalten.

11. Abschließende Betrachtungen / Lessons learned

Wichtig: In den abschließenden Betrachtungen ziehen Sie ein Projektfazit:

- War Ihre Aufwandsabschätzung aus Abschnitt 10 zutreffend?
 - o Welche Aufwände haben Sie zu Projektbeginn über- oder unterschätzt?
- Wie verlief die Teamarbeit insgesamt (positive/negative Aspekte)?
- Was würden Sie mit dem heutigen Wissen im nächsten Projekt anders machen?
- Was nehmen Sie an Positiven aus Ihrem Teamprojekt mit?
 - o Haben Sie bzgl. des professionellen Software-Engineerings neue Erkenntnisse gewonnen?
 - o Haben Sie bzgl. der Programmierung in einer höheren (objekt-orientierten) Programmiersprache neue Erkenntnisse gewonnen?

12. SW-Entwicklungsprozess nach Scrum (Anhang A, S. 231 ff.)

Soll das Produkt nach Scrum entwickelt werden, so ist nach der *Pre-Game-Phase* und Erstellung des *Product Backlogs* (inkl. Anforderungen und Grobdesign; dieses Dokument) das Produkt in folgenden *Sprints* (Entwicklungsphasen) zu erstellen:

1. Sprint: Im *Sprint Backlog* des 1. Sprints wird die Entwicklung einer rudimentären GUI nach Abb. 2 auf S. 3. festgelegt und im 1. *Sprint* vom Entwicklerteam entwickelt. Das *Sprint Review Meeting* des 1. Sprints soll dem Product Owner und Stakeholdern die generelle Funktionstüchtigkeit der App anhand eines Mockups inkl. einfacher Business Logik aufzeigen.
2. Sprint (inkl. Sprint Backlog und Sprint Review Meeting analog zu Sprint 1): Entwicklung des Corporate Designs der App (Look & Feel; Design Templates). Erweiterung der Business Logik nach Vorgaben.
3. Sprint: Entwicklung von exemplarischen statischen Unittests
4. Sprint: Erweiterung der Unittests mit Hilfe einer Testdatenbank und benutzergenerierten Testdaten

In der Post Game Phase wird die Dokumentation (Pflichtenheft, Benutzungshandbücher, Testdokumentation) erstellt. Zudem wird im Systemtest die Funktionstüchtigkeit des Systems nachgewiesen sowie im User Acceptance Test (Kundenabnahme) das System vom Kunden abgenommen.

Danach wird die Anwendung u.U. noch „deployt“, also beim Kunden installiert (In-House-Lösung) oder ein Link zur finalen Cloud-Lösung freigegeben.

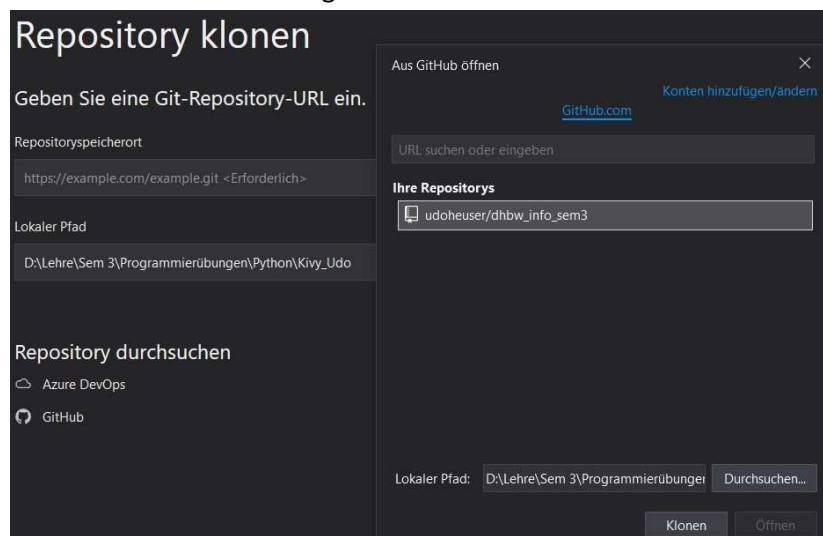
13. Anhang

a. Vollständige Python-Codes / GitHub

Die vollständigen Quell-Codes finden Sie unter dem GitHub-Verzeichnis:

https://github.com/udoheuser/dhbw_info_sem3

Diese Codebasis lässt sich mit Hilfe der Visual Studio IDE auf einfache Weise kopieren. Gehen Sie hierzu bitte auf Datei > Repository klonen > Repository durchsuchen: GitHub und geben danach die Download-URL von GitHub ein.



Weitere nützliche Tipps bzgl. GitHub:

- <https://de.github.com/features.html>
- <https://t3n.de/news/github-fuer-einsteiger-ersten-schritte-762760/>
- <https://www.dev-insider.de/das-erste-github-projekt-erstellen-a-670208/>
- https://youtu.be/H-AsOLO_sAk (Tutorial: GITHUB - DAS Tool für Developer)

Anmerkung: Typischerweise ist die Anzahl an Codezeilen bei größeren Softwareprojekten zu groß, um sie in ein Pflichtenheft hinzuzufügen – auch nicht im Anhang. In diesen Fällen hat es sich eingebürgert, lediglich Code-Snippets inkl. aussagekräftiger Code-Kommentare für besonders kritische Teile in das Pflichtenheft aufzunehmen, so wie in dieser Musterlösung angedeutet.

Die komplette Codebasis und sämtliche Dokumentation werden dabei auf GitHub (wie oben gezeigt), ownCloud oder sonstigen Cloud-Servern, welche frei zugänglich sind hochgeladen und der Link in das Pflichtenheft eingefügt.

So sollen also auch Sie in Ihrem Abschlussprojekt vorgehen: bei Unklarheiten oder weiteren Fragen kann so Ihr*e Dozent*in auf diese Quellen zurückgreifen.

- b. GitHub bietet neben der Dateiablage und Versionierung zudem weitere Möglichkeiten der Teamzusammenarbeit innerhalb VS. Hierzu muss GitHub als Extension zur VS hinzugefügt werden. Danach lassen sich Code-Änderungen per ‚push‘ an ein zuvor generiertes GitHub-Repository übertragen. Alle anderen Teammitglieder können diese Änderungen per ‚sync‘ mit ihren lokalen Code-Verzeichnissen abgleichen. Danach können sie am gemeinsamen Code weiterarbeiten. Mehr zur GitHub-Integration in VS erfahren Sie hier: <https://visualstudio.github.com/>, <https://www.lernmoment.de/alle/git-mit-visual-studio-2019/>
- Eine analoge Versionierung bietet auch Anvil integriert in seiner Web-IDE an: <https://anvil.works/docs/version-control-new-ide/quickstart>