Software Design

Julian Huber & Matthias Panny

State-Machines

“ ⚠️ Im Folgenden behandeln wir zunächst einfache State-Machines diese sind keine objekt-orientierten Behavioral Patterns im Sinne der Design Patterns ”

State-Machines

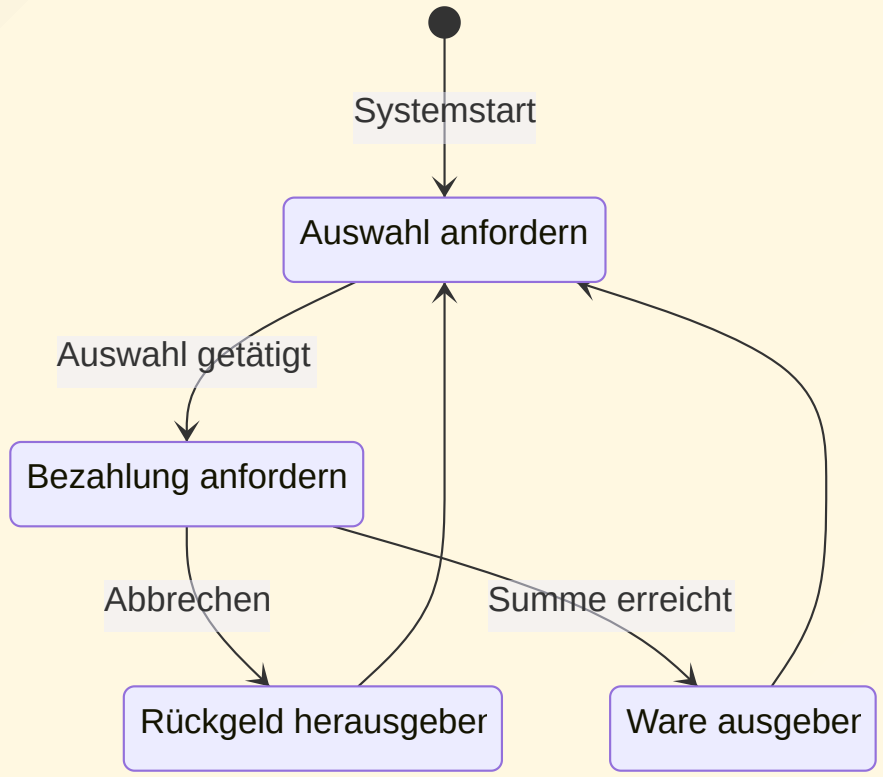
Herausforderung

- In Abhängigkeit vom Systemzustand erwarten wir unterschiedliches Verhalten z.B.
  - das GUI (Graphical User Interface) soll andere Dinge anzeigen
  - solange noch keine Geräte eingetragen sind, soll aufgefordert werden, ein Gerät anzulegen, ansonsten soll eine Liste der Geräte angezeigt werden
  - beim (Neu-)Start einer Software müssen Variablen deklariert und initialisiert werden sowie Pakete geladen werden
- Einfache Systeme lassen sich meist durch Ihre Zustände und deren Übergänge beschreiben
- Durch die Beschreibung als State-Machine können wir das Verhalten des Systems sauber trennen und die Implementierung vereinfachen

Beispiel

Implementierung eines Cola-Automaten

- Der Automat startet im Zustand `Auswahl anfordern`
- Der Automat kann in den Zustand `Bezahlung anfordern` wechseln
- Wird die geforderte Summe erreicht, wechselt der Automat in den Zustand `Ware ausgeben`
- Ist die Ausgabe abgeschlossen, wechselt der Automat wieder in den Zustand `Auswahl anfordern`
- Wird die Bezahlung abgebrochen, wechselt der Automat in den Zustand `Rückgeld herausgeben` und dann wieder in den Zustand `Auswahl anfordern`



State-Machines

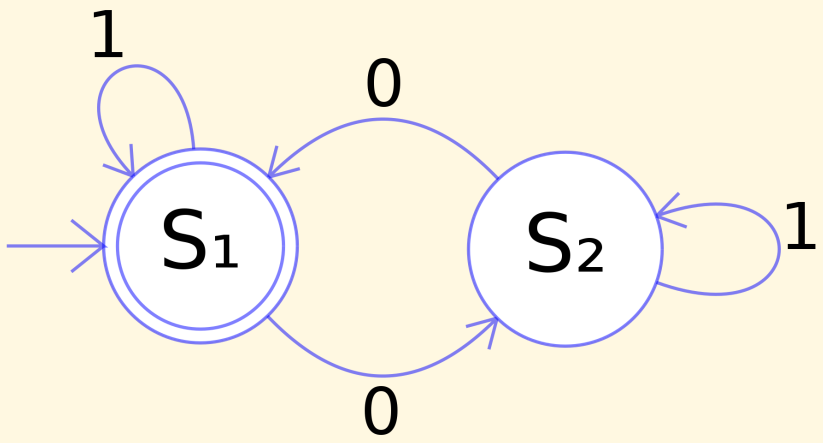
🤓 (Deterministic) Finite State Machine (Endlicher Automat)

- Dieser Ansatz basiert auf der Theorie der formalen Sprachen
- Eine DFSM beschreibt ein System mit endlich vielen Zuständen beschreibt
- Eine DFSM ist ein 5-Tupel $M = (Q, \Sigma, q_0, F, \delta)$ mit
  - Endlicher Zustandsmenge Q
  - Endliches Eingabealphabet Σ
  - Endlicher Startzustand $q_0 \in Q$
  - Endlicher Endzustandsmenge $F \subseteq Q$
  - Übergangsfunktion $\delta : Q \times \Sigma \to Q$

State-Machines

In der Theorie der formalen Sprachen 🤓

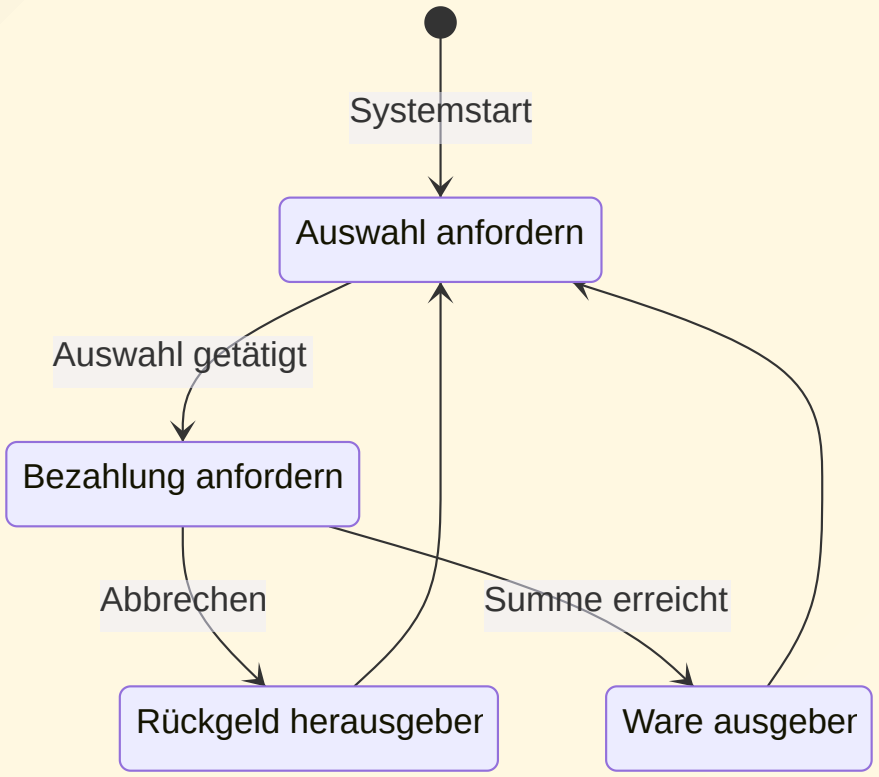
- $Q = \{S_1, S_2\}$
- $\Sigma = \{0, 1\}$
- $q_0 = S_1$
- $F = \{S_1\}$
- $\delta(S_1, 0) = S_2$, $\delta(S_1, 1) = S_1$
- $\delta(S_2, 0) = S_1$, $\delta(S_2, 1) = S_2$
- Akzeptiert (endet in F)
  - `1`, `11`, `01101`, `11001`
- Akzeptiert nicht (endet nicht in F)
  - `0`, `10`, `10100`, `01001`
- Anwendung
  - Parser, z.B. jede (wird durch) geschlossen
  - Regular Expressions



State-Machines

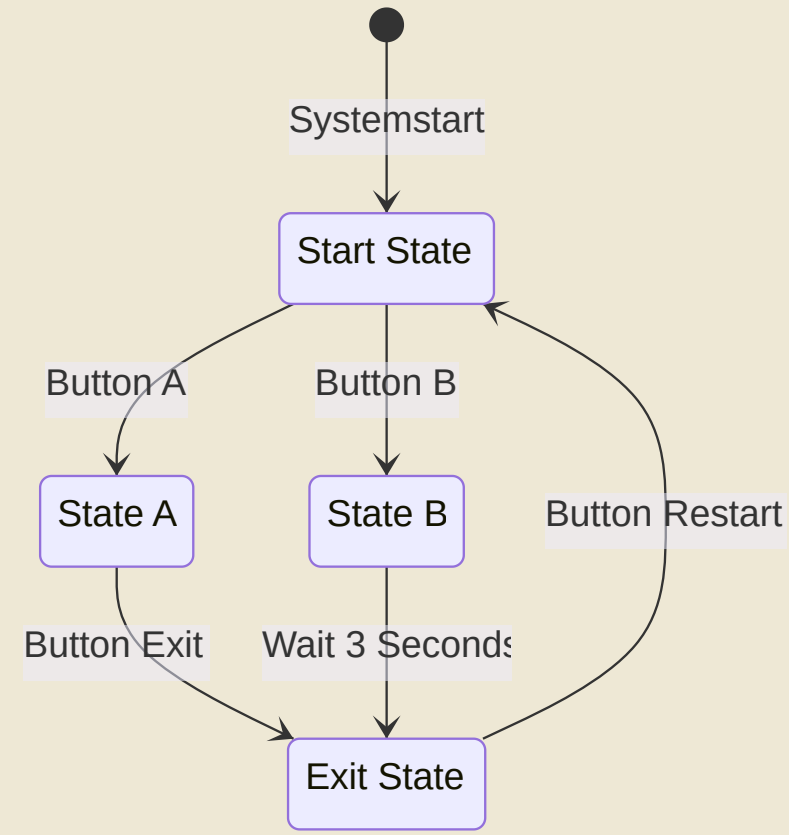
In der Programmierpraxis

- Die Knoten stellen Systemzustände dar. Innerhalb dieser Zustände muss das Systemen nicht statisch sein (z.B. kann während dem Systemzustand `Bezahlung anfordern` Münzgeld eingeworfen werden)
- Die gerichteten Kanten stellen Übergänge zwischen den Zuständen dar, die durch Events und Bedingungen ausgelöst werden
- Die Systemzustände können nur in festgelegten Reihenfolgen durchlaufen werden
- Ein Endzustand ist optional



Implementierung einer einfachen State Machine im UI

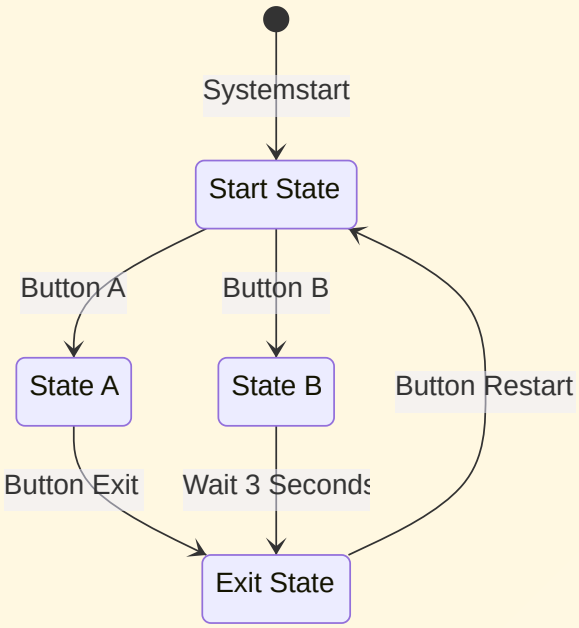
Beispiel `example_states.py`



Implementierung einer einfachen State Machine im UI

Beispiel `example_states.py`

- zunächst wird das Verhalten in den Zuständen definiert. Wir speichern den aktuellen Zustand in der Session-Variable `state`
- Da das `steamlit`-Skript bei jedem Aufruf neu ausgeführt wird, können wir die state variable so zwischen den Aufrufen speichern



Implementierung einer einfachen State Machine im UI

Beispiel `example_states.py`

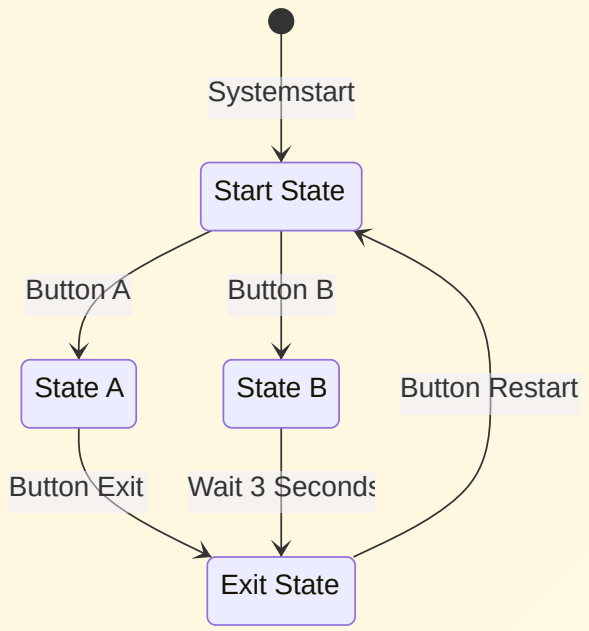
```python
import streamlit as st
import time

# Initialize state
if "state" not in st.session_state:
    st.session_state["state"] = "state_start"

elif st.session_state["state"] == "state_start":

    st.text("I'm in start state")
    # A callback function triggers a rerun of the script
    st.button("Go to state A!", type="primary", on_click= go_to_state_a)
    st.button("Go to state B!", type="primary", on_click= go_to_state_b)


elif st.session_state["state"] == "state_a":
    st.text("I'm in state A")
    st.button("Go to exit state!", type="primary", on_click= go_to_state_exit)

elif st.session_state["state"] == "state_b":
    st.text("I'm in state B")
    time.sleep(3)
    # With the rerun function we can rerun the script after a given time
    go_to_state_exit()
    st.rerun()


elif st.session_state["state"] == "state_exit":
    st.text("I'm in exit state")
    st.button("Restart!", type="primary", on_click= go_to_state_start)
```

Implementierung einer einfachen State Machine im UI

Übergänge zwischen den Zuständen

- Im State `state_b` wird nur ein Text angezeigt und nach 3 Sekunden wird der Zustand auf `state_exit` gesetzt
- die Funktion `go_to_state_exit()` setzt den Zustand auf `state_exit`
- mit `st.rerun()` wird das Skript neu ausgeführt

```
def go_to_state_exit():
    st.session_state["state"] = "state_exit"
...
elif st.session_state["state"] == "state_b":
    st.text("I'm in state B")
    time.sleep(3)
    go_to_state_exit()
    st.rerun()
```



Implementierung einer einfachen State Machine im UI

Übergänge mit Callbacks (Rückruffunktionen)

- Eine Funktion, die als Parameter einer anderen Funktion übergeben wird
- Die übergebene Funktion wird dann innerhalb der anderen Funktion aufgerufen, wenn bestimmte Bedingungen erfüllt sind (z.B. Button geklickt)
- Häufig in Graphischen Benutzeroberflächen verwendet und durch das Framework implementiert (im Fall von `streamlit` wird dabei automatisch das Skript neu ausgeführt)

```
def go_to_state_exit():
    st.session_state["state"] = "state_exit"

...
elif st.session_state["state"] == "state_a":
    st.text("I'm in state A")
    st.button("Go to exit state!", type="primary", on_click=go_to_state_exit)
```

Callbacks (Rückruffunktionen)

Beispiel

- Callbacks sind nicht auf UI-Elemente beschränkt, sondern können überall verwendet werden

```
def main_function(x, callback):
    # Perform some operation
    result = x * 2

    # Call the callback function with the result
    callback(result)

def callback_function_a(result):
    print(f"The result is: {result}")

def callback_function_b(result):
    print(f"THE RESULT IS: {result}!!!")

# Example usage
main_function(5, callback_function_a)
main_function(5, callback_function_b)
```

Callbacks (Rückruffunktionen)

Verwandte Design Patterns

- *Inversion of Control*: Die Kontrolle über den Programmablauf wird an eine andere Komponente übergeben, welche nur dazu da ist zu entscheiden, wann welche *Callback-Funktion* aufgerufen wird. Die Callback-Funktionen selbst implementiert was gemacht wird
- Da der Programmablauf nicht mehr zentral vom Server gesteuert wird, wird von einer umgekehrten Kontrolle gesprochen
- Ein Beispiel ist auch der *Listener* (Beobachter-Muster). Hierbei wird ein Systemzustand überwacht (z.B. wurde mit dem User Interface interagiert) und bei gewissen Events eine Callback-Funktion aufgerufen

State Machine

Mikrocontroller - microcontroller_sm.c

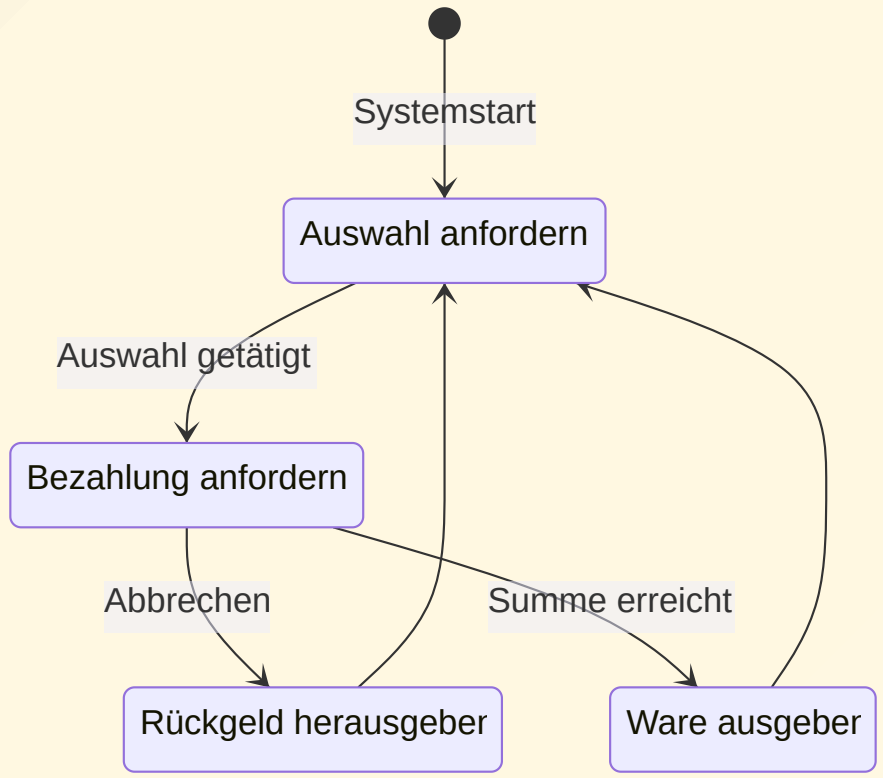
- In der Programmierung von Mikrocontrollern werden State Machines häufig verwendet
- Die State Machine wird in einer Endlosschleife ausgeführt

```c
#include <stdio.h>
#include "timing.h" //header that contains fictitious get_time_ms()

typedef enum State { ST_START, ST_STATE_A, ST_STATE_B, ST_STATE_EXIT } State;

State state = ST_START;
int start_wait_b;

void main(){
    start_wait_b = get_time_ms();
    while(1){
        switch(state){
            case ST_START:
                printf("I'm in start state - Loading initial data\n");
                state = ST_STATE_A;
                break;
            case ST_STATE_A:
                printf("I'm in state A - Sending data somewhere\n");
                state = ST_STATE_B;
                start_wait_b = get_time_ms(); //Reset wait time for state B
                break;
            case ST_STATE_B:
                printf("I'm in state B - Waiting without blocking\n");
                if(get_time_ms() - start_wait_b > 30000){
                    state = ST_STATE_EXIT;
                    start_wait_b = get_time_ms();
                }
                break;
            case ST_STATE_EXIT:
                printf("I'm in exit state - exiting gracefully\n");
                state = ST_START;
                break;
        }
    }
}
```

✍️ Aufgabe: UI eines Cola-Automaten

Aufgabenstellung

- Implementieren Sie die State Machine eines Cola-Automaten in `streamlit`
- (einfach): Nutzen Sie als Grundlage die State Machine aus `example_states.py` und verwenden Sie nur `st.button` und `st.text`
- (herausfordernd) nutzen sie Sie UI-Elemente aus `cola_without_states.py` um ein realistischeres UI zu erstellen



✍ Aufgabe: UI eines Cola-Automaten

Musterlösung `cola_with_states_simple.py`

- Wo weicht die Implementierung von der Grafik ab?
- Warum ist diese Anwendung nur schwer zu warten?

Musterlösung `cola_with_states.py`

- Was könnte man hier verbessern?

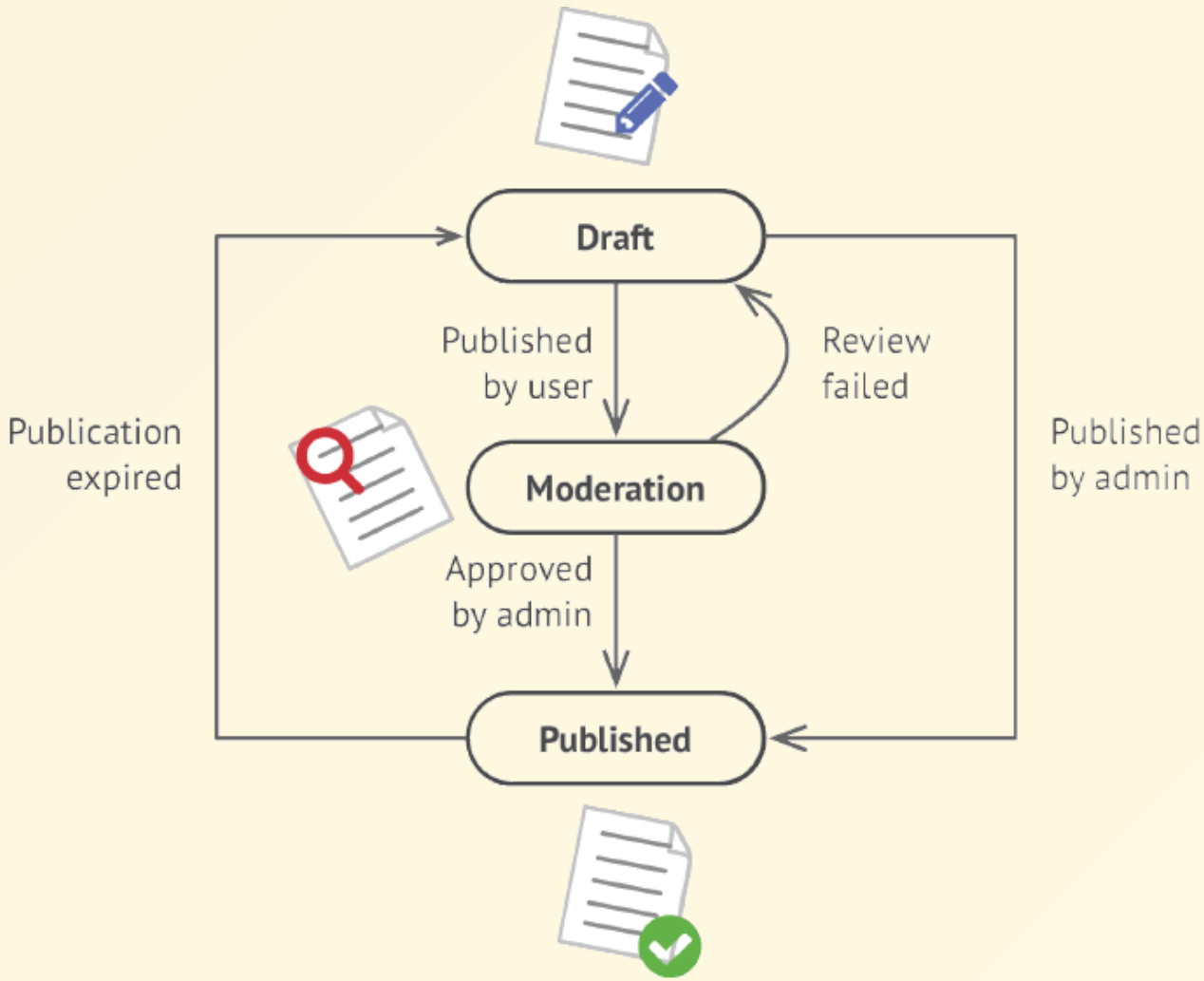
State Pattern

(als Beispiel für ein Behavioral-Design-Pattern)

State Pattern

In der OOP

- Es gibt auch ein State Pattern in der Objektorientierten Programmierung
- Hierbei ändert sich das Verhalten eines Objekts, wenn sich sein interner Zustand ändert
- Siehe Shvets 2019



Possible states and transitions of a document object.

State Pattern

Beispiel `state_pattern_document_naive.py`

- Naiver Ansatz: mit `if`-Abfragen den Zustand des Dokuments abfragen und das Verhalten ändern
- die `publish()`-Methode ändert das Verhalten des Dokuments in Abhängigkeit vom Zustand und dem Benutzer-Typ
- die Conditionals können schnell sehr komplex werden

```
class Document:
    def __init__(self):
        self.state = "draft"

    def publish(self, current_user):
        if self.state == "draft":
            if current_user.role == "user":
                self.state = "moderation"
            elif current_user.role == "admin":
                self.state = "published"
        elif self.state == "moderation":
            if current_user.role == "admin":
                self.state = "published"
        elif self.state == "published":
            pass  # Do nothing.
```

State Pattern

Beispiel `state_pattern_document.py`

```python
from abc import ABC

class State(ABC):
    @abstractmethod
    def publish(self, document, current_user):
        pass

class Draft(State):
    def publish(self, document, current_user):
        if current_user.role == "user":
            document.state = Moderation()
        elif current_user.role == "admin":
            document.state = Published()

class Moderation(State):
    def publish(self, document, current_user):
        if current_user.role == "admin":
            document.state = Published()

class Published(State):
    def publish(self, document, current_user):
        pass  # Do nothing.

class Document:
    def __init__(self):
        self.state = Draft()

    def publish(self, current_user):
        #self == the current document!
        self.state.publish(self, current_user)
```