# Software Engineering

Cicio Ionuț

22/12/2024

# Contents

# 1 Software models

When designing **complex software** we have to make major **design choices** at the beginning of the project. Often those choices can't be driven by experience or reasoning alone, that's why a **model** of the project is needed to compare different solutions. Our formal tool of choice is the **Discrete Time Markov Chain** (Section 1.2.3).

## 1.1 The "Amazon Prime Video" article

If you were tasked with designing the software architecture for **Amazon Prime Video** *(a live streaming service for Amazon)*, how would you go about it? What if you had the **non-functional requirement** to keep the costs as low as possible?

In a recent article, Marcin Kolny, a Senior SDE at Prime Video, describes how they *"**reduced the cost** of the audio/video monitoring infrastructure by 90%"* [1] by using a monolith application instead of distributed microservices (an outcome one wouldn't usually expect).

While there isn't always definitive answer, one way to go about this kind choice is building a model of the system to compare the solutions. In the case of Prime Video, *"the audio/video monitoring service consists of three major components:"* [1]

- the **media converter** converts input audio/video streams
- the **defect detectors** execute algorithms that analyze frames and audio buffers in real-time looking for defects and send notifications
- the **orchestrator** controls the flow in the service

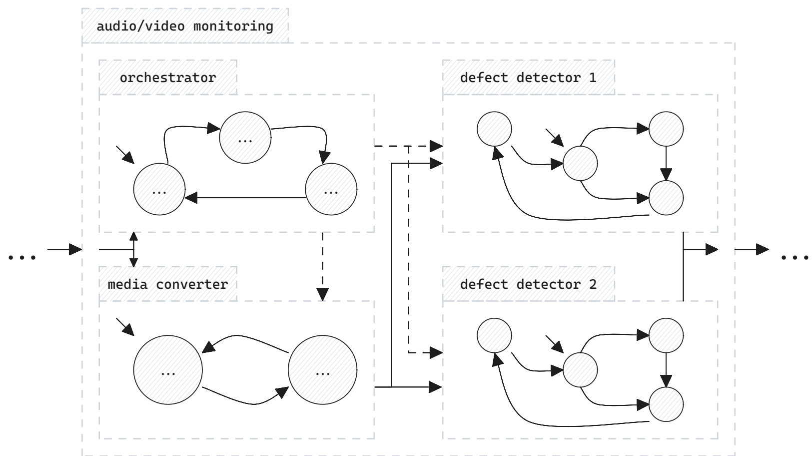


Figure 1: Model of the audio/video monitoring system

We want to model the components as some kind of stateful machine (with inputs and outputs, Figure 1), interconnect them and **simulate** the behaviour of the system as a whole.

## 1.2 Formal notation

### 1.2.1 The concept of time

TODO: rewrite

The models treated in the course evolve through **time**. Time can be modeled in many ways (I guess?), but, for the sake of simplicity, we will consider discrete time. Let $W$ be the *'weather system'* and $D$ the *'driving ability' system* in Section 1.2, we can define the evolution of $D$ as

$$\begin{aligned} &D(0) = \text{'good'} \\ &D(t+d) = f(D(t), W(t)) \end{aligned} \tag{1}$$

Given a time instant $t$ (let's suppose 12:32) and a time interval $d$ (1 minute), the driving ability of $D$ at 12:33 depends on the driving ability of $D$ at the time 12:32 and the weather at 12:32.

### 1.2.2 Markov Chain

A Markov Chain is...

### 1.2.3 DTMC (Discrete Time Markov Chain)

A DTMC $M$ is a tuple $(U, X, Y, p, g)$ s.t.

- $U \neq \varnothing \wedge X \neq \varnothing \wedge Y \neq \varnothing$ *(otherwise stuff doesn't work)*
- $U$ can be either
  - $\{u_1, ..., u_n\}$ where $u_i$ is an input value
  - $\{()\}$ if $M$ doesn't take any input
- $X = \{x_1, ..., x_n\}$ where $x_i$ is a state
- $Y = \{y_1, ..., y_n\}$ where $y_i$ is an output value
- $p : X \times X \times U \to [0, 1]$ is the transition function
- $g : X \to Y$ is the output function

$$\forall x \in X \;\; \forall u \in U \quad \sum_{x' \in X} p(x'|x, u) = 1 \tag{2}$$

$$\begin{aligned} &M(0) = x_1 \\ &M(t+d) = \begin{cases} x_1 & \text{with probability} \;\; p(x_1|M(t), U(t)) \\ x_2 & \text{with probability} \;\; p(x_2|M(t), U(t)) \\ \ldots \end{cases} \end{aligned} \tag{3}$$

It's interesting to notice that the transition function depends on the input values. If you consider the *'driving ability'* system in Section 1.2, you can see that the probability to go from `good` to `bad` is higher if the weather is rainy and lower if it's sunny.

#### 1.2.3.1 An example of DTMC

Let's consider the development process of a team. We can define a DTMC $M = (U, X, Y, p, g)$ s.t.

- $U = \{()\}$, as it doesn't have any input
- $X = \{0, 1, 2, 3\}$
- $Y = \text{Cost} \times \text{Duration (in months)}$

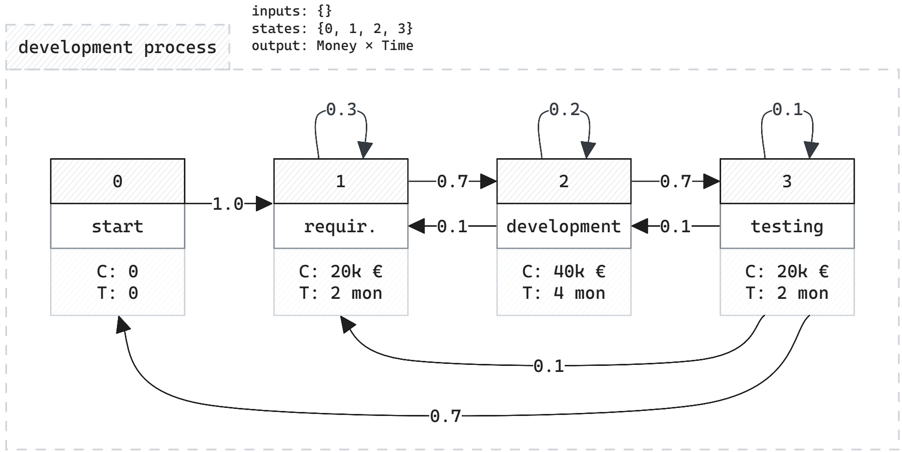


Figure 2: the model of a team's development process

$$g(x) = \begin{cases} (0, 0) & \text{if } \ x = 0 \\ (20000, 2) & \text{if } \ x = 1 \\ (40000, 4) & \text{if } \ x = 2 \\ (20000, 2) & \text{if } \ x = 3 \end{cases} \tag{4}$$

### 1.2.4 Network of Markov Chains

TODO...

## 1.3 Tips and tricks

### 1.3.1 Mean

TODO: 'mean' trick, ggwp

$$\varepsilon_n = \frac{\sum_{i=0}^{n} v_i}{n}$$

$$\varepsilon_{n+1} = \frac{\sum_{i=0}^{n+1} v_i}{n+1} == \frac{\left(\sum_{i=0}^{n} v_i\right) + v_{n+1}}{n+1} = \frac{\sum_{i=0}^{n} v_i}{n+1} + \frac{v_{n+1}}{n+1} = \tag{5}$$

$$\frac{\left(\sum_{i=0}^{n} v_i\right) n}{(n+1)n} + \frac{v_{n+1}}{n+1} = \frac{\sum_{i=0}^{n} v_i}{n} \cdot \frac{n}{n+1} + \frac{v_{n+1}}{n+1} = \frac{\varepsilon_n \cdot n + v_{n+1}}{n+1}$$

### 1.3.2 Eulero's method for differential equations

Useful later...

# 2 C++

## 2.1 Intro to #include <random>

### 2.1.1 Seed & std::default_random_engine

### 2.1.2 Distributions

#### 2.1.2.1 std::uniform_int_distribution<>()

#### 2.1.2.2 std::uniform_real_distribution<>()

#### 2.1.2.3 std::bernoulli_distribution<>()

#### 2.1.2.4 std::poisson_distribution<>()

#### 2.1.2.5 std::geometric_distribution<>()

#### 2.1.2.6 std::discrete_distribution<>()

## 2.2 Intro to data structures

### 2.2.1 std::vector<T>()

### 2.2.2 std::deque<T>()

### 2.2.3 Sets

### 2.2.4 Maps

## 2.3 I/O

### 2.3.1 #include <iostream>

### 2.3.2 Files

# 3 Exercises

Each exercise has 4 digits `xxxx` that are the same as the ones in the `software` folder in the course material.

## 3.1 First examples (`1000`)

Now we have to put together our **formal definitions** and our `C++` knowledge to build some simple DTMCs and networks.

### 3.1.1 A simple Markov Chain (`1100`)

Let's begin our modeling journey by implementing a DTMC $M$ s.t.

- $U = \{()\}$ it takes no input
- $X = [0, 1] \times [0, 1]$ it has infinite states (all the pairs of real numbers between 0 and 1)
- $Y = [0, 1] \times [0, 1]$
- $p : X \times X \times U \to X = \mathcal{U}(0, 1) \times \mathcal{U}(0, 1)$
- $g : X \to Y : (r_0, r_1) \mapsto (r_0, r_1)$ it outputs the current state
- $X(0) = (0, 0)$

```cpp
#include <fstream>
#include <random>

using real_t = double;

int main() {
    std::random_device random_device;
    std::default_random_engine random_engine(random_device());
    std::uniform_real_distribution<real_t> uniform(0, 1); ①

    const size_t HORIZON = 10; ②
    std::vector<real_t> state(2, 0); ③

    for (size_t time = 0; time <= HORIZON; time++)
        for (auto &r : state)
            r = uniform(random_engine); ④

    return 0;
}
```

Listing 1: `software/1100/main.cpp`

### 3.1.2 Connect Markov Chains pt.1 `(1200)`

In this exercise we build 2 markov chains, and connect them…

Now let's model a system with two DTMCs $M_0, M_1$, and lets define the functions

$U(M_0, t+1) = \cdots \; U(M_1, t+1) = \cdots$

### 3.1.3 Connect Markov Chains pt.2 `(1300)`

The same as above, but with a different connection

### 3.1.4 Connect Markov Chains pt.3 `(1400)`

The same as above, but with a different connection

## 3.2 Traffic light `(2100, 2200, 2300)`

This is

## 3.3 Control center `(3000)`

### 3.3.1 No network `(3100)`

### 3.3.2 Network monitor (no faults) `(3200)`

### 3.3.3 Network monitor (faults, no repair) `(3300)`

### 3.3.4 Network monitor (faults, repair) `(3400)`

### 3.3.5 Network monitor (faults, repair, correct protocol) `(3500)`

## 3.4 Statistics `(4000)`

### 3.4.1 Expected value `(4100)`

### 3.4.2 Probability `(4200)`

## 3.5 Transition matrix

## 3.6 Complex systems

### 3.6.1 Insulin pump

### 3.6.2 Buffer

### 3.6.3 Server

# 4 Exam

## 4.1 Development team (time & cost)

## 4.2 Backend load balancing

## 4.3 Heater simulation

# Bibliography

[1] Marcin Kolny, “Scaling up the Prime Video Audio-Video Monitoring Service and Reducing Costs by 90%.” Accessed: Mar. 25, 2024. [Online]. Available: https://web.archive.org/web/20240325042615/https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90#expand