Table of Contents

Project Plan

Introduction

Technology is ever-evolving so the constant effort to follow agreed-upon practices is not enough without an understanding of the context you are developing within. There is no right or wrong just the most convenient solution. To understand why we do things this way one must dive into the project characteristics which is what we will do next.

The 4 Ps:

- Product → the software to be built
- People → a crucial element of a successful project
- Process → the set of framework activities and software engineering tasks to get the job done
- Project → all work required to make the product a reality

PRODUCT:

API to serve as the backend for several types of client interfaces in the cinema chain. This software is to be built from the ground up; there is no code base or other larger system to be integrated within. The data coming from the client is to be saved in the database and then returned on a format agreed upon with the product owner, as well as the nature and specifications for the requests the API serves.

PEOPLE:

- Customer → the product owner
- End-user → customers of the cinema theater and cinema employees

We have no interaction with the end-user, all our communication is with the Product Owner, so we are absolved of any domain analysis. Our focus is to deliver a reliable back-end for the requirements given.

- Practitioners → 3 computer science students

Based on the given product and the team of practitioners and since this is an academic project we came with a structure that will allow each of us to take turns into one of the following roles to face the responsibilities and come with its analysis after fulfillment of the role in one given sprint.

a. *Project manager*: In charge of analysis (risk analysis, holds the retrospective meetings, and given results he calculates the velocity)
b. *Scrum master*: In charge of setting deadlines and division of work (based on experience or an analysis of the complexity of the tasks and the inter-dependencies), holds daily meetings, solves conflicts, gives estimates.
c. *Programmers*: all together

As a team we have not collaborated before, so we need to have enough meetings to know our style, our strengths, and our weaknesses. To produce the above team structure, we took into consideration some factors like:

- the difficulty of the project is at a medium level since we are familiar with the development of this type of project (client-server application, java world technology stack, and an agile way of doing things) yet still, we might face challenges that could put the delivery of the project at risk so a project manager that will follow the risks and the overall development of the project is required.
- the size of the project would be around 1500–2000 lines of code (calculation made based on previous projects of this size and type), so we need a person in charge of keeping an eye on the cleanness and uniformity which is the scrum master.

- the project can be modularized so simultaneously work is possible after an initial agreement on the data model, and the means to make this project reality.
- we have no budget for a more performant server so at first glance response times from the server will be bigger than normal which results in a somewhat poor user experience.
- the date to deliver the product is fixed and cannot be changed so risks must be managed to deliver at least a viable product.

PROCESS:

A Software Process Model is a description of the sequence of activities carried out in a software engineering project and the relative order of these activities. The models should be aids to thinking, not rigid prescriptions of the way to do things.

We were given a strict order of the most important requirements by the Product Owner and requirements are frozen until the next iteration (domain analysis is assumed to have been carried out by the product owner).

In this scenario, the best fit for our needs is an agile approach following the incremental model to develop already fixed requirements and build on top of them with each iteration.

Once the development of an increment has been initiated the requirements are frozen, so our only focus is on the specific feature which was agreed upon with the product owner.

To this we can add that at the base of our software engineering process we were striving to follow the Layered Technology pyramid starting with:

a Quality focus:

- documented endpoints (using Swagger)
- work conducted accordingly with DevOps principles (early CI/CD pipeline using Jenkins)
- separation of development/production environment (Maven profiles)
- a reliable database with frequent backups in the local environment as well as a permanent database online
- and a relentless try to follow the TDD (Test Driven Development) principles.

the Process Model:

- the development methodology to be used for this project is an agile-iterative way of doing things that relates to the well-known *Incremental Model*.

In this case, what fits best our way of doing things was the *XP framework* of activities and *Scrum* as the way to manage our team.

➔ *XP* delivers the practices that ensure high quality
➔ *Scrum* manages the process.

XP practices that we applied:

- rapid communication with the customer
- a common vision of how the program works
- simple design
- short releases
- coding standards
- collective ownership
- continuous integration.

An argument for our choice would be that *Scrum* is widely used to manage *XP* projects and gives a significant deal of autonomy and with an *Agile way* of doing things risk is just business risk and not overall project risk.

Communication, Tools, and Motivation:

Communication is done on multiple channels. Firstly, we have a scrum board (we used Trello) and a git repository that is central to what we are doing, then we hold daily meetings on Teams or in person if possible, and a messenger chat for immediate actions.

Every person has an agreed task to work upon and is free to organize their work as long as the deadline is respected.

We come up with promising ideas by giving each other the possibility to speak up, take charge in solving problems, and help each other when in doubt. (self-organizing team)

Taking all these factors into consideration we made a courageous assumption that we can safely achieve our goals in the time given.

PROJECT

Answering:

- *What work should be done?*
- *In what sequence to perform the work?*

Calculating estimates *(an attempt to determine how much money, effort, resources, and time it would take to build a specific software system*)

requires:

- experience
- access to good historical information (metrics)

and depends on the:

- project complexity
- project size

Estimation techniques used by us:

- to determine the size of the project: *Function Points*
- to make estimates: scrum master *complexity analysis*

Function Points

Measurement Parameter	Count	Weighting factor			Total
		Simple	Average	Complex	
Number of user inputs	**7**	3	4	6	**28**
Number of user outputs	**16**	4	5	7	**80**
Number of user inquiries	**9**	3	4	6	**27**
Number of files	**1**	7	10	15	**10**
Number of external interfaces	**0**	5	7	10	**0**
Count					**145**

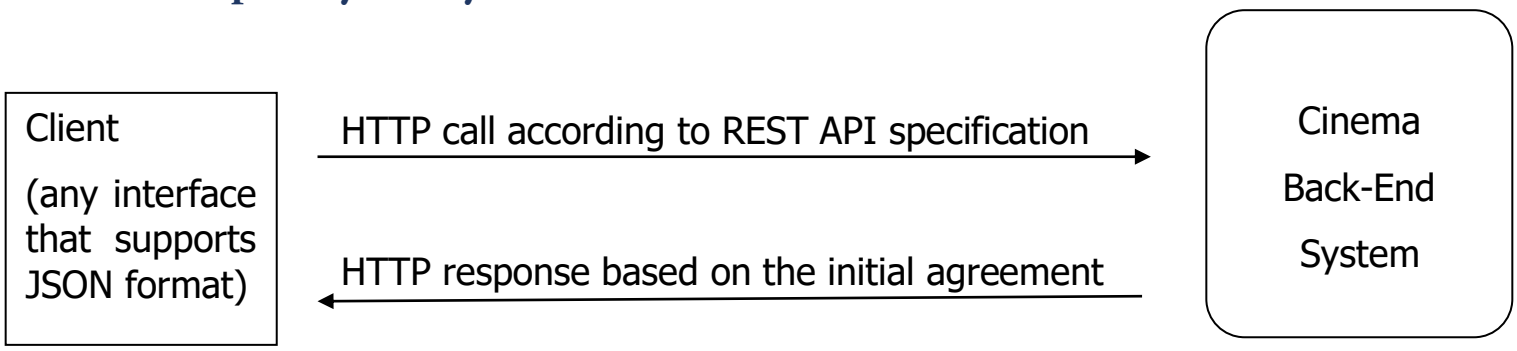
14 Factors of Complexity: from 0 (not important) to 5 (essential)

Back-up and recovery – **5**	Heavily used configuration – **3**	Complex input – **3**
Data communication – **5**	Online data entry – **5**	Designed for reuse – **5**
Distributed functions – **3**	Transaction rate – **3**	Installation included – **3**
Performance – **4**	Complex processing – **3**	Multiple installations – **3**
Online update – **5**	Facilitate change – **5**	**TOTAL**: **55**

Function Points = count x [0.65 + 0.01 x complexity factors)]

= 145 x (0.65 + 0.01 x 55) = 174

Complexity Analysis



Cinema employee APIs:

1.REST API for CRUD Movie

2.REST API for CRUD Movie Playing

3.REST API for CRUD Show

Customer APIs:

1.REST API to search for movie playing on date

2.REST API to search for movie playing between dates

3.REST API to book ticket for show

All the REST API calls follow a similar pattern, so division is possible to work efficiently assuming that we have agreed on the format of the movie class and have some initial data to be able to test the other APIs.

EXAMPLE**:** REST API for CRUD Movie

Phase One → Define data and repositories.

1. Design DTOs for the input and output data required by client (parameters have been defined in the requirements)
2. Create Java Classes that support these DTOs
3. Create Java repositories that connect to the database and extract data based on the attributes of the above defined classes

Complexity level: *EASY*

Phase Two → Create the controllers and services that return a response when controller calls them.

1. Endpoint to receive data (REST API controller in Java)
2. Validation and duplication checks (create the service for this)
3. Persist data into a database (create the service for this)
4. Create a response for the client (create the service for this)
5. Send data (controller sends the service response)

Complexity level: *MEDIUM*

Scheduling principles:

- compartmentalization and defined responsibilities:
  - well defined and distinct tasks are to be assigned (after a complexity analysis)
- interdependency:
  - when assigning tasks, we will take into consideration using a burndown chart to manage dependencies
- time allocation:
  - is to be given by a mutual decision with the developer to receive the task based on his current skills and available time
- defined outcomes following testing principles:
  - unit testing, service testing, integration testing

Early Timebox:

Task Name	Duration
1. Preliminary Design	**2 days**
1.1 Data model	1 day
1.2 System architecture	1 days
2. Implementation	**5-6 days**
2.1 First User Story	1 day
2.2 Second User Story	1-2 days
2.3 Third User Story	1 day
2.4 Fourth User Story	2 days
Total	**1 week**

Umbrella activities

Focus is on risk management and technical reviews between programmers to assure the quality for a minimum viable product.