# Project Task

Project goal is to deliver versatile recommender engine. Many modern websites are having their content tailored for their users. It keeps their attention and time spent on the platform significantly increases. From customer perspective we would like to achieve recommendations for the user that is looking for entertaining movies to watch. From our perspective it means that we should use available data and possible solutions to meet expected requirements.

# Understand the problem

## a. Who has a stake in the solution to the problem? That is, who are the stakeholders?

First stakeholder would be end user. We rely on data that is created by the user activity on the website. We must ensure that data is free of fake records and application is malware free. Users have tendency to exaggerate their feels and attitude toward movies. Their opinion is biased by our feelings. They love or naturally dislike some cast members. They appreciate some movie directors more than others. Such scenarios are complicating correct verdicts.

Second stakeholder are developers. Recommendation engine should have been built with accordance to available data. Correct approach may maximize its correctness.

## b. What are the unknowns? What data, functions, and features are required to solve the problem properly?

The Unknown could be lack of data. Correct engine handles possible scenarios. Has its limitations. For example, how recommendations should be calculated when user has no data? Possible solutions are:

* Simple and quick surveys about user’s preference.
* Lists based on friends that are currently using application.
* Features that become available when recommendation system results are reliable.

Required data is users activity on the website. In basic recommendations it would be sufficient to use just customers and their ratings on particular movies.

Advanced systems are tracking much more user activity. For example Netflix tracks users activity during watching movies or tv series. When do they make a pause. How many episodes they tend to watch at once etc.

## c. Can the problem be compartmentalized? Is it possible to represent smaller problems that may be easier to understand?

One of principles in problem solving is to break it into smaller parts. The problem is to build working recommender engine system. Smaller issues might be:

* Verifying data
* Choosing software to build it.
* Choosing algorithm.
* Verifying results.
* Handling errors.

## d. Can the problem be represented graphically? Can an analysis model be created?

ASK POMOROVA

# Plan a solution

This activity isolates requirements and develops both size and resource estimates. Also, a defect estimate (the number of defects projected for the work) should be made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified, and a project schedule is created.

## a. Have you seen similar problems before? Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?

Both of us have experience in using such feature. Starting from Youtube movies recommender that out of nowhere recommends video that has been uploaded even few years ago. If it works fine it can easily attract people in spending more time on their website than user planned.

Another example would be Spotify. This music streaming application relies on recommendations even more. They provide user daily mix of music that they have in library and adds new tracks between them. Application itself might be very useful for music producers. Spotify can verify producers music and provide information about who actually listens to them.

Last big example would be Netflix. They track every user activity. As the result of such big data they can entertain people on their platform. They can decide what types of tv series would be interesting for people. Based on that they started to create their own tv series. Just to fill the gap.

These are the biggest examples within entertainment industry. They have been working with such problems for years. They had to adopt their ideas. They slowly evolved and some of them had to make an revolution in recommendation’s engine industry. They use bigger and more complex and advanced solutions. However, origin stays the same and without user’s data and correct implementation it would not been possible to solve such idea.

## b. Has a similar problem been solved? If so, are elements of the solution reusable?

Such problems have been solved. Elements of one implementation might be used once again (if similar environment exists). Reusability is created by architecture of the application. Loosely coupled design, delegated services, dependency injection and we have ready to use and share library.

## c. Can subproblems be defined? If so, are solutions readily apparent for the subproblems?

Subproblems can be defined. For example formulas consist of smaller calculations. It allows to focus on data correctness and accuracy. Implementing formula by ourselves indicates possible variations. Moreover, we started to wonder what approach would fit best.

## d. Can you represent a solution in a manner that leads to effective implementation? Can a design model be created?

Our idea is to not only implement real formula into code and display results. We would like to have working prototype of real application. We would like to test different approaches. With back-end, front-end, database. We would like to measure how long it would take to calculate all needed predictions. Whether these can be done “Just in time” or it should be done “ahead of time”.

We are able to create initial design model. Although, it requires verification whenever we are making decision and major change in our application design.

## e. List of Requirements. Describe Requirements for your Task. Use CHECKLIST end Examples from Lab 2.

Requirements Modeling Principles

Principle #1. The information domain of a problem must be represented and understood.

Principle #2. The functions that the software performs must be defined.

Principle #3. The behavior of the software (as a consequence of external events) must be represented.

Principle #4. The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.

Principle #5. The analysis task should move from essential information toward implementation detail. f. Create UML Diagrams and DESCRIBE them. 1. Use-case diagrams; 2. Sequence diagrams; 3. Collaboration diagrams; 4. Activity Diagrams. 5. Class diagram.

4. Carry Out the Plan. External specifications for each component to be constructed are developed and component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked. Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results. The component-level design is refined and reviewed. The code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.

3

a. Does the solution conform to the plan? Is the source code traceable to the design model? b. Is each component part of the solution provably correct? Has the design and code been reviewed, or better, have correctness proofs been applied to the algorithm? c. Describe SOLID Design Principles for your Project + examples. d. On the base of Lecture 6 – last slide – Task for Lab 9:

# Examine the Result

Using the measures and metrics collected (this is a substantial amount of data that should be analyzed statistically), the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness. a. Is it possible to test each component part of the solution? Has a reasonable testing strategy been implemented? b. Does the solution produce results that conform to the data, functions, and features that are required? Has the software been validated against all stakeholder requirements? c. Analyze YOUR Classes: Describe Reasons to create for each Class in your Project. d. Analyze YOUR Classes: Verify your Classes according CHECKLIST: Class Quality. e. Analyze YOUR Classes: Describe Classes which does not satisfy CHECKLIST and possible causes of emergence such classes. f. Analyze YOUR design (Project software) for its overall conformance to the Main Sequence. The D metric for each component can be calculated. Any component that has a D value that is not near zero can be re-examined and restructured. g. Chose the Static Code Analyser for your project and analyze source code. Describe your typical errors.

# Include your Presentation Screenshots (4 slide per page).

# Conclusions.