# Product Planning

Team: goto fail;

Alex Geenen
Martijn Janssen
Mark van de Ruit
Bart de Jonge
Menno Oudshoorn

# Table of Contents

## Introduction

In this document, we will give a broad overview of how we plan to develop our product over the next 9 weeks. We will discuss which features will, and will not be implemented. Also, we will give a schedule of what features will be released in which week. We will also have a look at some user stories of users that will be using our product, and how they affect the product backlog. Also, we'll be looking into a definition of done, to make sure every project member knows exactly when they can call a feature, sprint, or release, “Done”.

## High-level product backlog

We will use the MoSCoW method to define the features we do and do not implement. The MoSCoW method divides features into four categories:

1. Must haves. These are features that must be implemented for the product to work correctly. Without these features, there will be no release
2. Should haves: These are features that are not directly fundamental to the product working, but enhance the quality of the product greatly and therefore, a project team should aim to implement as much of these features as possible
3. Could haves: These are features that are not something that one directly thinks of as needed when thinking about the product, but are nice enhancements to possibly make in the future, if the time allows it
4. Won't haves: These are features that will not be implemented, or at least not in the first version of the product

The three main sub-products we will be creating are a script creator, an interface for automatic camera preset recalling, and interfaces to show the timeline during actual recording. We will give MoSCoW features for each of these products separately.

### Script creator

- Must haves
  - Intuitive loading and saving of created scripts.
  - Timeline interface to show all camera shots in order, and per camera.
  - Intuitive adding and deleting of camera shots.
  - Moving camera shots around through intuitive drag-and-drop interface.
  - Highlighting of shots that collide with other shots.
  - Director timeline that shows a summary of camera shots.
  - Ability to change size of timeline, relative to number of counts.
- Should haves
  - Automatically generated camera timelines from constraints specified by the director.
  - Ability to add camera to multiple timelines at the same time.
  - Intuitive editing of camera shots
- Could haves
  - Modularity and customizability of GUI.
  - Editing.
- Won't haves
  - None for now.

### Camera preset recalling

- Must haves
  - Intuitive interface to store and recall presets
  - A web server used to operate the cameras
  - Automatic preset recalling during recordings

    - Possibility to switch off automatic recall and take over manual control
- Should haves
    - Live low-quality preview of upcoming cameras.
- Could haves
    - Possibility to adjust the cameras through the preset interface.
- Won’t haves
    - Live high-quality preview of upcoming cameras in our application

## Timeline interfaces during recording

- Must haves
    - Different views for different people, such as the director, a camera operator, the score caller
    - Ability to easily move ahead through the timeline.
    - Ability for the director to change the timeline on the fly.
    - Ability for the score caller to change the speed at which the timeline progresses.
- Should haves
    - Ability for the score caller to choose between manual and automatic timeline speed, and change this within a recording.
    - Highlighting those cameras that a certain camera operator is operating
    - Warnings for possible problems when the director edits the timeline during recording, such as collisions.
- Could haves
    - Ability to tap the timeline to see more information about a shot
- Won’t haves
    - None for now.

## Roadmap

In the roadmap, we will give a brief overview of the features we want to implement every week. It is based on scrum principles, and the focus lies on having a working version every week. This means that every week's sprint builds on the working version from the week before. The roadmap is as follows:

| Week | Features |
|---|---|
| 4.2 | Create mockup of GUI of script creator<br>GUI first version with drag and dropping<br>Product vision and product planning<br>Initial data model and I/O<br>Basic collision detection algorithm for shots. |
| 4.3 | Basic GUI styling<br>Saving and loading projects<br>Collisions in GUI. Detection and styling<br>Adding and deleting shots<br>Editing shots (length, description, etc.)<br>Settings screen<br>Basic director's timeline |
| 4.4 | Director able to set up basic ideas and constraints<br>Automatic creation of camera timelines from director constraints<br>Setting up web server for camera control |
| 4.5 | Experimenting with IP commands to control cameras<br>Creating mock/initial interface for creating/storing presets<br>Create mocks for views during recording |
| 4.6 | GUI for live timeline views (score caller, director, camera operators)<br>Initial automatic preset recall algorithm . |
| 4.7 | Live camera preview in preset interface<br>Allow live timeline changes and propagate these to preset recaller. |
| 4.8 | Time to finish everything that couldn't get done in previous weeks |
| 4.9 | Finalize features, fix possible bugs.<br>Final design changes<br>Fixing all static analysis warnings<br>Exporting product |

## User stories

In this section, we will give some user stories that show the preferences of the user. These preferences should align with the features given earlier. We will divide the user stories into three categories: director, camera operator and score caller. These all have different needs of our product.

### User stories for the director

As a director, I would like to be able to easily create scripts using an intuitive interface. This interface should allow me to save and load scripts with ease.

As a director, I would like to be able to easily modify scripts using an intuitive interface. The interface should allow me to place, move and remove shots with ease.

As a director, I would like to be able to easily modify shots in a script using an intuitive interface. The interface should allow me to change types, lengths and other variables of shots.

As a director, I would like to be able to specify a set of constraints with respect to which cameras could be used in a possible shot, and let the separate camera timelines automatically.

As a director, I would like to be visually notified when I make a script that has collisions or other potential problems.

As a director, I would like to be able to change the script on the fly, to adapt to unexpected situations.

### User stories for a camera operator

As a camera operator, I would like to be able to easily see what shots I have to take for the next number of counts, for all my cameras, in one small intuitive interface.

As a camera operator, I would like to have more time to focus on my actual shots, so I want my next presets to be loaded automatically.

As a camera operator, I would like to be able to see a live preview of my upcoming shots, so I can already think about possible adjustments.

As a camera operator, I would like to be able to take over manual control at all times, for when an unexpected situation arises.

### User stories for the score caller

As a score caller, I would like to be able to easily see what shots are coming up and how long it takes before they come up.

As a score caller, I would like to be able to choose between automatic timeline speed, or a manual timeline in which I have to tap a button for each count.

As a score caller, I would like to be able to take over manual control at all times, for when an unexpected situation arises.

## Definition of done

In this section, we will show what is considered as the definition of done within our project team. This is done so that every project member has the same idea about when something can be called “Done”, and this should make sure that no unfinished features or products are handled as if they are done.

The definition of done is divided into three parts: features, sprints and release. We will discuss the definition of done for each of these three elements

### Features

- Code is written according to standards, no Checkstyle errors
- Unit tests have been written for the feature, and pass
- Acceptance criteria of the user story are met
- Functional tests are performed by a team member who did not work on the feature
- All existing tests still pass after merging with the stable branch

### Sprints

- All the items in the sprint backlog have been implemented, and all these items are done as specified in above definition
- All code is written according to standards, no Checkstyle errors
- Relevant documents (such as architecture design) have been created or updated
- Sprint retrospective has been created
- All existing tests pass

### Release

- All the specified Must have features have been implemented
- A significant amount of Should have features have been implemented
- All tests pass
- No checkstyle errors
- Final product is well documented
- Customer must agree on the final result